Research Report

Version 1.0

The Trickle Algorithm: Issues and Solutions

Badis Djamaa and Mark Richardson

Centre for Electronic Warfare, Cranfield University,
Defence Academy of the United Kingdom, Shrivenham SN6 8LA
{b.djamaa, m.a.richardson}@cranfield.ac.uk

20 January 2015

The Trickle Algorithm: Issues and Solutions

Badis Djamaa and Mark Richardson

Abstract

To manage the various multi-purpose Internet of Things applications brought about by low-power and lossy networks, efficient methods of network configuration and administration; firmware installation and updates; neighbourhood, route and resource discovery are required. These requirements can be reduced to a basic data consistency maintenance problem, making the Trickle algorithm a powerful candidate solution. Trickle is shaped by the so-called short-listen problem, hence the imposition of a listen-only period. Such a period allows Trickle to robustly address the short-listen problem at the expense of increased latency. In this report, we revisit the Trickle rules, the short-listen problem and interval-synchronisation, and hence introduce New-Trickle. New-Trickle is an optimisation to Trickle with virtually no extra cost in terms of communication overhead, computation demand and implementation effort, yet one that provides fast updates, yielding a propagation time more than 10 times faster than Trickle.

1. Introduction

Wireless Sensor Networks (WSNs) have been recognized as a key technology in shaping tomorrow's world [1]. Their applications have been significantly widened by the emerging Internet of Things (IoT) vision which aims to integrate sensors and actuators, computers and PDAs, smartphones and various other components in a global architecture in order to get the most out of them [2]. Thus, IoT foresees billions of sensors and actuators connected to the internet in heterogeneous contexts, ranging from home and building automation to assisted living and health monitoring. This large number of low-power constrained nodes internetworked via lossy links brings many challenges to protocol and application design. The computing, memory, communication and power constraints characterizing such Low-power and Lossy Networks (LLNs) necessitate simple best-effort solutions that reduce and balance energy consumption. At the same time, user and/or application requirements envisage adequate quality of service (QoS) in terms of reliability, responsiveness and timeliness.

To address the above (generally conflicting) requirements and manage the diverse multi-purpose IoT applications of WSNs, efficient methods of network configuration and administration; firmware installation and updates; neighbourhood, route and resource discovery are required. Such methods need to rely on simple, reliable and efficient mechanisms that can quickly and consistently propagate new information (e.g. commands, configurations, sensed events) inside/to/from the network while keeping the generated overhead, hence the energy consumption, as low as possible. This is so, in order to minimize congestion and maximize the network lifetime. These requirements can be reduced to a basic data consistency maintenance problem, making the Trickle algorithm [3] [4] a very powerful candidate to solve a wide range of problems, including but not limited to the above. For instance, Trickle[1] can ensure that the nodes involved in a routing structure have consistent information, as is the case of its usage in the IPv6 Routing Protocol for LLNs (RPL) [5] and the Collect Tree Protocol (CTP) [6].

The rationale behind Trickle is to provide LLN nodes with a simple, local, robust, energy-efficient, and scalable information exchange primitive. Trickle achieves this by relying on two basic mechanisms: adaptive transmission periods and a timer-based suppression mechanism. Dynamically adjusting transmission windows to the network context allows Trickle to achieve quick resolution of inconsistencies while sending few consistency control packets when the network is in a steady state. On the other hand, a simple timer-based suppression mechanism allows Trickle's communication rate to scale logarithmically with network density. Thus, a node using Trickle suppresses its own transmission if made aware that neighbouring nodes are spreading the same message. Finally, Trickle is simple to implement and only requires few resources in terms of memory and computational power [4].

The aforementioned Trickle characteristics make it a very attractive basic network primitive for many LLN applications. However, despite the abundant work related to the Trickle algorithm, and to our knowledge, no extensive analysis and experimentation of Trickle behaviour [4] has been published. This motivated us to carry out the present work in order to provide such an analysis. Indeed, we introduce a simple, yet very powerful optimisation to Trickle that can

[1] Trickle with a capital T is used in what follows to mean: The Trickle Algorithm

dramatically decrease its inconsistency resolution time while preserving its scalability. To this end, we begin with an in-depth analysis of Trickle, its advantages and drawbacks in section 2. This is followed by introducing the proposed optimisation, its basic concept and main benefit in section 3. Section 4 is devoted to demonstrating that such an optimisation does not affect Trickle's robustness and scalability, although it might introduce a small extra overhead. Expected other benefits of the proposed New-Trickle are presented in section 6. The evaluation methodology of New-Trickle when compared to Trickle and Short-Trickle (a version of Trickle suffering from the short-listen problem) is detailed in section 6. Section 7 discusses extensive results obtained from cycle-accurate simulations and large-scale public testbed experiments. This is followed by the impact of New-Trickle in Section 8. Related works are the subject of section 9. This report ends with conclusions and discussions of future directions.

2. The Trickle Algorithm

2.1. Overview

The Trickle algorithm [3][4] has gained much popularity and emerged as a basic network primitive that can ensure fast and reliable resolution of data inconsistencies with low maintenance cost, thereby saving energy and minimizing network congestion, while scaling well with network density. Trickle, documented as an internet standard in RFC 6206 [4], is deployed in two other internet standards [5][7] and is being used in many applications such as routing control traffic [6], distributed service/resource discovery [8] and reliable broadcast/dissemination [9][10][11][12]. For instance, routing protocols such as RPL [5] and CTP [6] use Trickle in order to manage routing control traffic frequency. The IPv6 Multicast Protocol for LLNs (MPL) [7] being currently standardized by the Internet Engineering Task Force (IETF) heavily rely on Trickle to achieve reliable multicast in LLNs. Thus, MPL relies on Trickle in both its proactive and reactive modes. Furthermore, Trickle is the state of the art algorithm used in dissemination and over-the-air programing protocols in WSNs. It is the heart of Deluge [9], Dip [10], Drip [11] and DHV [12]. Moreover, Trickle is becoming a basic networking primitive in LLNs [13], and it is delivered as a standard library in major WSN operating systems such as TinyOS[2] and Contiki OS[3].

A node using Trickle periodically broadcasts its data unless it has recently heard identical data. As long as nodes agree on what data they have, they increase their transmission periods exponentially. When data disagreements are detected, Trickle starts transmitting more quickly. To realise this behaviour, and as per [4]'s notation, the Trickle algorithm maintains three variables; namely:

- A consistency counter c.
- A Trickle interval I.
- A transmission time t within an interval I.

In addition, Trickle defines three configuration parameters, namely:

- The minimum interval size $Imin$ ($Imin$ is defined in units of time, e.g., milliseconds, seconds).
- The maximum interval size $Imax$ (*Imax* is described as a number of doublings of $Imin$ and hence the time specified by $Imax$ would be $Imin \times 2^{Imax}$). Note that for brevity, we sometimes use $Imax$ to mean the time specified by $Imax$.
- The redundancy constant k (k is an integer greater than zero, a value of zero has a specific meaning of infinity).

When Trickle starts, it sets the consistency counter c to zero, the interval size I to a random value between $[\mathrm{Imin};\ \mathrm{Imin} \times 2^{\mathrm{Imax}}]$ and picks the transmission time t uniformly at random from $[\mathrm{I}/2; \mathrm{I})$. Whenever a node hears identical data (dotted lines in Figure 1), it increments c. At time t, a node transmits (dark box in Figure 1) if and only if c is less than k. Otherwise, the transmission is suppressed (grey line in Figure 1). When the interval I expires, Trickle doubles the interval length until the time specified by Imax, from which the interval length is kept fixed. Finally, if a node hears inconsistent data and I is greater than $Imin$, I is set to Imin. Otherwise, Trickle does nothing. Whenever I is set (a new interval begins), c is reset to zero and t to a random value in $[\mathrm{I}/2; \mathrm{I})$. As per RFC 6206, the aforementioned Trickle behaviour can be expressed by the following six steps:

- ***Step* 1**: When Trickle starts, it picks uniformly at random I form the range $[Imin;\ Imin \times 2^{Imax}]$ and begins the first interval.
- ***Step* 2**: When an interval begins, Trickle resets c to 0 and picks t uniformly at random from the range $[\mathrm{I}/2; \mathrm{I})$.

[2] https://github.com/tinyos/tinyos-main/blob/master/tos/lib/net/TrickleTimerImplP.nc
[3] https://github.com/contiki-os/contiki/blob/master/core/lib/Trickle-timer.c

- ***Step* 3**: Whenever Trickle hears a *consistent* transmission, it increments the consistency counter c.
- ***Step* 4**: At time t, Trickle transmits if and only if the consistency counter c is less than the redundancy constant k.
- ***Step* 5**: When the interval I expires, Trickle doubles the interval length until the time specified by $Imax$. Trickle then starts a new interval as in ***Step* 2**.
- ***Step* 6**: If Trickle hears an *inconsistent* transmission while I is greater than $Imin$, it resets the Trickle timer. To do so, Trickle sets I to $Imin$ and starts a new interval as in ***Step* 2**. Otherwise, i.e. I was equal to Imin when an *inconsistency* is detected, Trickle does nothing. Note that the Trickle timer can also be reset by external *events*.

Leaving the terms *consistent*, *inconsistent*, and *events* generic allows Trickle to be adapted and adopted by various protocols that interpret their meanings.

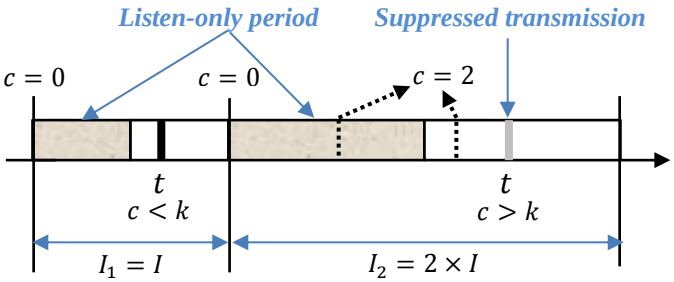
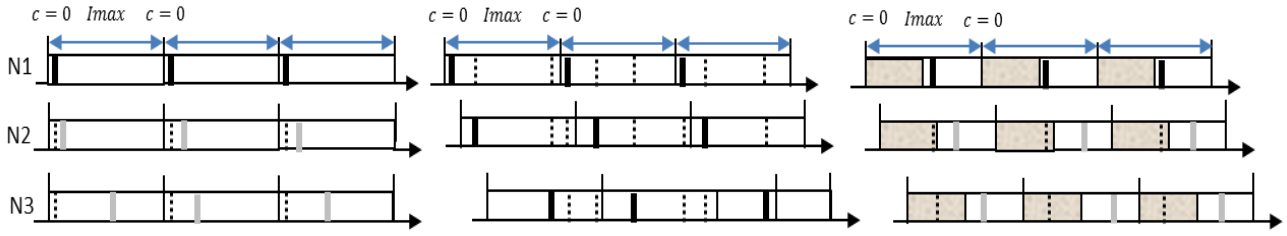


Figure 1 Trickle algorithm over two intervals with $k = 1$. Black line is a transmission, grey one is a suppressed transmission and dotted lines are receptions

A noticeable point in the Trickle rules (particularly Step 2), which is depicted in Figure 1, is the so-called listen-only period spreading over the first half of each Trickle interval, hence dividing it into two main parts: a listen-only part and a transmission period. Indeed, this listen-only period is introduced in response to a challenging problem to Trickle called the short-listen problem, discussed in [3]. The short-listen problem occurs mainly because of non-synchronised Trickle intervals between neighbouring nodes. It has a drastic impact on Trickle's suppression mechanism, thereby on Trickle's scalability. Figure 2 (a) and (b) illustrate the impact of the short-listen problem on Trickle's suppression mechanism in a single-hop network comprising three nodes. Figure 2 (a) shows the expected efficiency of the suppression mechanism when node intervals are tightly synchronised. Thus, even when considering a worst-case where the random transmission time selection process makes N1 transmit at the beginning of every interval, nodes N2 and N3 are able to catch this transmission before transmitting own data and hence can suppress their redundant transmissions. This is achieved thanks to interval synchronisation, which allows Trickle to scale with the expected k transmission per interval in a lossless network. However, if node intervals are not synchronised (Figure 2 (b)), no node can hear N1's transmission before its own, hence nodes N2 and N3 keep competing to transmit. Suppose that N2 listens only for a short time before it transmits, then N3 will not be able to hear such a transmission before its own and hence decides to transmit, which makes the suppression mechanism useless in this particular case. As this problem is caused by nodes choosing to listen for short periods before deciding to transmit, it is dubbed short-listen problem in [3].



(a) Synchronized network **(b) Short-listen problem** **(c) Listen-only period**

Figure 2 Short-listen problem and listen-only period with k = 1. Black boxes are transmissions, dotted lines are receptions and grey boxes are suppressed transmissions.

In the general case, [3] shows that the short-listen problem causes the number of transmissions in a single interval to scale as $O(\sqrt{n})$ (n being the number of nodes in a single-hop lossless network), instead of k in synchronised lossless networks or the aimed at $O(\log(n))$ in lossy networks. Getting to synchronize node intervals and maintain synchronisation between them is a resource-consuming task that can be very expensive in terms of memory, communication and computing demands. Furthermore, even if node clocks can be synchronised, there is no guarantee that Trickle intervals can be too. Indeed, besides losses, Trickle itself can cause node intervals to be non-synchronised. A simple, stateless and yet powerful solution to this problem is to impose a listen-only period at the start of each interval. In this period, a node only listens for incoming messages. Such a period has shown to bound the generated number of messages per interval by a constant inversely proportional to the size of the listen-only period [3]. However, a bigger listen-only period might have a dramatic impact on the propagation time as it delays a transmission by at least the length of the listen-only period at each hop. Opting for a fair time/cost trade-off, Trickle defaults to a listen-only period of a half-interval (Step 2), which asymptotically bounds the number of generated transmissions per interval by $2 \times k$ in lossless networks [3]. As can be seen from Figure 2 (c), the default listen-only period allows nodes N2 and N3 to suppress their transmissions even if the random transmission time selection process makes N1 transmits just at the end of its listen-only period. This brings Trickle's scalability to the desired $O(\log(n))$ in the general case of lossy networks.

The final shape of the Trickle algorithm was arrived at by years of experimenting and running Trickle across various application domains. Based on which, RFC 6206 [4] advises to not try to tweak Trickle's behaviour without providing extensive experimental evidence. Indeed, the authors of RFC 6206 state *"Based on our experiences, we urge protocol designers to suppress the instinct to tweak or improve Trickle without a great deal of experimental evidence that the change does not violate its assumptions and break the algorithm in edge case"*. Nevertheless, Trickle in its actual final format suffers from many issues, some of which are caused by the listen-only period. The following subsection discusses the main ones.

2.2. Criticisms

As shown above, the listen-only period allows Trickle to scale logarithmically with network density at the expense of increased delays. Such delays have the most impact when resolving inconsistencies, as they postpone every transmission by at least the size of the listen-only period, which is half of an (Imin) interval. This makes the introduced delay heavily dependent on the value of Imin, further aggravating the latency in networks adopting relatively large Imin values, such it is the case of Trickle usage in distributed service discovery protocols [8]. Additionally, this Imin-dependent delay gets accumulated at every hop in multi-hop networks, which results in a considerable latency for a packet travelling long distances (in terms of hops). This issue is the main driver behind our work, as it introduces considerable delays to our Trickle-based service discovery protocol [8].

Added to the aforementioned main issue, the listen-only period might introduce unbalanced transmission loads in the network, making some nodes transmit more than others in some intervals. This might not affect the overall load-balancing performance of Trickle, as its inherent mechanisms and well-design can compensate for temporary unbalanced load distribution. However, the listen-only period can explicitly prevent some critical nodes from transmitting, which might further delay network consistency. This compound issue is illustrated in Figure 3 and Figure 4. Focusing on the load-balancing issue, Figure 3 depicts the Trickle intervals of three neighbouring nodes receiving an update from different senders. As can be seen from this figure, node N1 has the biggest chance to transmit in the $Imin$-interval compared to nodes N2 and N3. Indeed, N3 has zero-chance to transmit as its listen-only period is totally overlapped with the transmission period of N2. Hence N3's transmission is explicitly prevented by the listen-only period, not by the suppression mechanism (more on this in the following paragraph). From the next interval (I_1 in Figure 3), the chances of N2 and N3 to transmit increase and keep increasing with increased interval sizes. For instance, in the fourth interval, the three nodes have similar transmission chances. To sum up, network dynamics can cause $Imin$ intervals of contending neighbours to be non-synchronised. The amount of non-synchronisation between these intervals is aggravated by the listen-only period. This non-synchronisation implies an unbalanced transmission load distribution, which have most impact on smaller intervals, especially on $Imin$-sized ones. It can even cause a more serious problem - to be discussed in the following paragraph. Fortunately, once a new data appears this situation may change and hence will not get infinite. In addition, the chances for nodes to transmit become closer and closer as intervals size increase, as shown in Figure 3. This suggests that choosing bigger values of $Imax$ helps distribute Trickle's transmission load equitably between nodes.

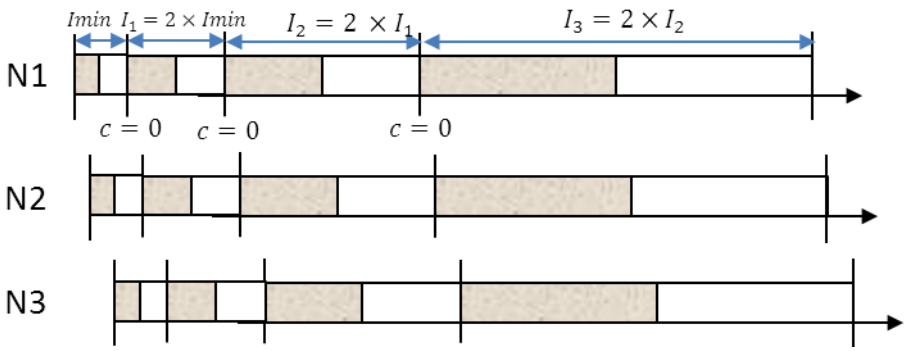
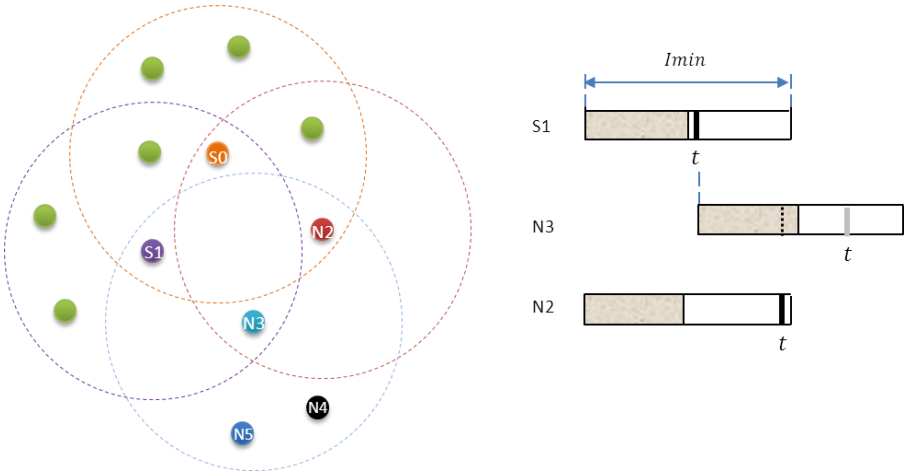


Figure 3 Trickle load balancing issue

Back to the other issue caused by the listen-only period, illustrated in Figure 3, which prevents N3 from transmitting. This issue is illustrated via a detailed example of a multi-hop lossless network, depicted in Figure 4 (a). When a seed node S0 transmits an update, all its direct neighbours, i.e. the nodes situated inside the circle centred at S0, (S1 and N2 being two of them), shrink their intervals to Imin. At the end of its listen-only period, S1 transmits the received update to its neighbours. S1's neighbours which receive the update for the first time (N3 being one of them) will shrink their intervals to Imin. Other S1's neighbours hearing the re-transmission will simply suppress own ones. Before transmitting, N3 has to wait for at least the length of the listen-only period, which entirely overlaps with N2's transmission period, hence forcing N3 to suppress its transmission (Figure 4 (b)). This stops the update from reaching N4 and N5 in this interval, delaying them by at least another Imin time (which is the length of the next interval listen-only period), with fewer chances for N3 to transmit compared to N2 and S1. This might postpone updating N4 and N5 to the following intervals. This problem is more visible in sparse networks or networks containing irregularities and holes than in dense networks. It might also become worse if the applications are using smaller Imax doubling; such is the case of MPL's proactive mode. Finally, it should be noted that opting for a $k > 1$ can minimize the probability of this problem appearing. However, it adds to Trickle's cost.



(a) (b)

Figure 4 The listen-only period preventing nodes from transmitting ($k = 1$).

A third criticism emerges from the fact that the listen-only period cuts the transmission part in a Trickle interval by half (from I to I/2), causing nodes to contend only in the second half of the interval. This may impact the efficiency of the suppression mechanism, especially in dense networks, since many nodes contend in smaller intervals. This can increase the probability of collisions and chances of hidden-terminals. This effect is especially important when the size of the interval is small. Indeed, protocols opting for small Imin values might be impacted more than others, as it was observed in our experiments. The bigger the size of an interval, the better the suppression efficiency of a timer-based suppression mechanism using uniform random selection points - such it is the case of Trickle's suppression mechanism. It should be noted that combining this issue with the previous one can help Trickle to minimise the cost generated in smaller intervals.

Finally, while the listen-only period robustly addresses the short-listen problem, it does not guarantee the optimal transmissions per interval of k in a lossless single-hop network. Instead it bounds this number by $2 \times k$ when using a half interval listen-only period [3].

In the following sections, we propose a simple, yet very powerful optimisation to Trickle that addresses our criticisms (especially the main one concerning the propagation time), discuss its impact on Trickle's cost and scalability, and finally demonstrate its benefits in respect of the other points discussed above. This is done keeping in mind the recommendations of RFC 6206 to provide *"a great deal of experimental evidence"*.

3. The New-Trickle Algorithm

Having presented an overview of the Trickle algorithm, explained its rationale and discussed our criticisms, we introduce in this section a simple, yet powerful optimisation to Trickle, which gives birth to the New-Trickle Algorithm (New-Trickle).

3.1. The proposed optimisation

Our optimisation is based on a basic observation from $Step\ 6$ of the Trickle algorithm. $Step\ 6$ of Trickle triggers the nodes receiving an inconsistency to immediately (assuming that receptions occur simultaneously) start new intervals of size Imin (if $I > Imin$). This can present an implicit synchronisation of $Imin$-sized intervals between these nodes, which comes at no cost and exactly when needed. Such a synchronisation can allow these nodes to choose I from [0; Imin) without experiencing a short-listen problem with each other. Based on this observation, we propose to modify $Step\ 2$ of the Trickle algorithm as follows:

- ***Step* 2**: When an interval begins, Trickle resets c to 0 and picks t uniformly at random from the range:
  - $[0;\ Imin)$, if the interval began as a result of ***Step* 6** (because of an *inconsistency* or in response to external *events*).
  - $[I/2; I)$, otherwise (the interval began as a result of ***Step* 1** or ***Step* 5**).

Note that neighbours can experience non-synchronised Imin-sized intervals as a result of losses and/or the multi-hop nature. Fortunately, an implicit synchronisation in the transmission periods of these intervals remains valid, as will be detailed in section 4. However, there is no guarantee of implicit synchronisation in the following intervals, and hence the listen-only period is deployed.

3.2. How this Makes Trickle Propagate Faster

As Trickle resolves inconsistencies in $Imin$-sized intervals, the proposed optimisation is expected to drastically decrease the propagation time of Trickle at virtually no extra cost. On first glance, it can be thought of the propagation time to be halved. However, many parameters (e.g., $Imin$ value, network density, hop count) can influence the propagation time, allowing it to be much faster, as will be seen and discussed in the results section. For the sake of giving an estimate, below are some observed consistency time factors obtained from evaluating Trickle and New-Trickle in a 400-node network deployed in a 20x20 grid. Multi-hop results are from a dense deployment of about 36 neighbours per node and a network diameter of about 13 hops. The default value of k was one.

- In a single-hop, very lossy network, New-Trickle propagated about 11 times faster than Trickle when using an $Imin$-value of two seconds.
- Using an $Imin$ value of two seconds, New-Trickle propagated around seven times faster than Trickle in a physically lossless dense multi-hop network.
- With the previous configuration, New-Trickle decreased the propagation time by a factor of 3.5, when using an $Imin$ value of one second.
- The New-Trickle algorithm propagated more than twice faster than Trickle in a very lossy dense multi-hop network using an $Imin$ of one second.

Having briefly presented and shown the principal benefit of New-Trickle, we will focus in the following section on its detailed conceptual basis and challenging issues, and demonstrate through in-depth analysis its impact on Trickle's scalability.

4. Does New-Trickle Preserve Trickle's Scalability

In this section, we discuss New-Trickle's scalability and demonstrate that it preserves Trickle's logarithmic scalability. To this end, we start with a simple case of a single-hop lossless network. Then, we relax this assumption by looking at multi-hop lossless networks and then by introducing losses in single- and multi-hop networks. Note that no assumptions are made about node interval synchronisations, and hence we opt for the general case of non-synchronised node intervals. New nodes joining the network are implicitly included in this analysis. Without loss of generality, a Trickle redundancy constant of one ($k = 1$) is assumed in the following analysis.

4.1. Lossless, single-hop networks

When a node N1 propagates an update in a single-hop lossless network, all other nodes will receive it and, by Trickle's *Step* 6, immediately start new Imin-sized intervals (i.e. at the *"same"* time). This can be considered as an implicit-synchronisation between these nodes. Hence, the short-listen problem is not be experienced by these nodes if they choose t from $[0;\ \mathrm{Imin})$, as shown in Figure 5. Note, however, that whichever receiver transmits (e.g. N2, N3 or N4 in Figure 5) in the *Imin* interval, it might experience a short-listen with the second interval of the originator (N1). The impact of this does not affect Trickle's scalability, as will be discussed in section 4.5.

This idealistic case shows the basic conceptual building block behind the proposed optimisation. It also clearly demonstrates that the listen-only period of the *Imin* interval grows the interval shift between neighbours, for instance, between the originator (N1) and the other nodes depicted in Figure 5.

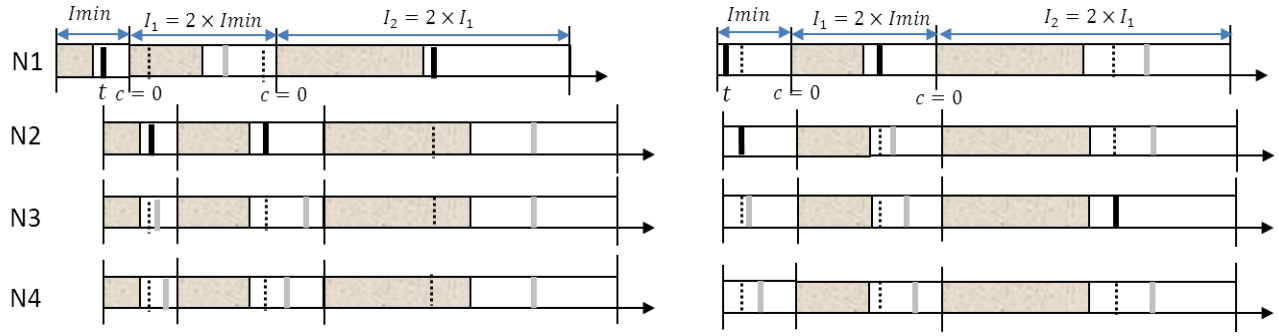


Figure 5 Trickle (left) and the New-Trickle (right). The grey rectangle represents the listen-only period. Black lines are transmissions, grey lines are suppressed transmissions and dotted lines are receptions.

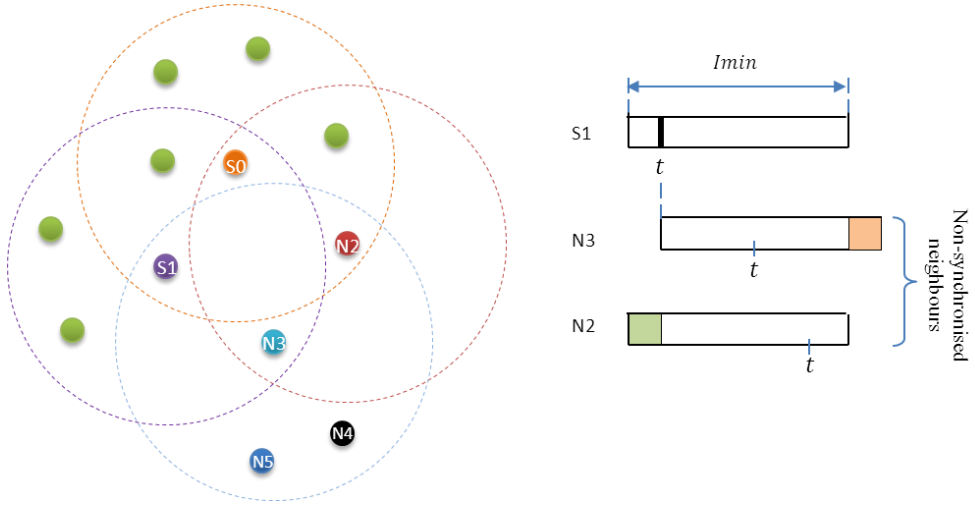
4.2. Lossless, multi-hop networks

This subsection examines lossless, multi-hop networks. Without loss of generality, we express this case with an example depicted in Figure 6 (a) (similar to the one presented in Figure 4 for Trickle). Suppose that a seed node S0 has D direct neighbours. As the network is lossless, all the D nodes (S1 and N2 being two of them) shrink their intervals to Imin when receiving S0's update. New-Trickle allows such nodes to choose t from $[0;\ \mathrm{Imin})$ as they are implicitly synchronised. Suppose now that S1 is the first to retransmit the update. S1's neighbours that receive the update for the first time (N3 being one of them) will shrink their intervals to *Imin* and thereby can choose t from $[0;\ \mathrm{Imin})$ without experiencing a short-listen problem between each other. Other S1's neighbours hearing the retransmission simply suppress their transmissions.

The aforementioned process can result in non-synchronised Imin-sized intervals between nodes N2 and N3, which are neighbours competing to propagate the update, as shown in Figure 6 (b). This challenges the implicit synchronisation between the *Imin* intervals of neighbours, observed in the previous idealistic case. Nevertheless, because N2 is still competing to transmit, meaning it did not transmit in the past part of its interval (the green rectangle in Figure 6 (b)), the beginning of the transmission periods of both N2 and N3 are implicitly synchronised. Therefore, either N2 or N3 transmits first; the other transmission will be suppressed and hence no short-listen problem would be experienced by either N2 or N3. Finally, it should be noted that as the network is lossless, N3 also cannot transmit in the orange part of its *Imin*

interval when $k = 1$ (Figure 6 (b)). This is because N2 would have transmitted before the end of its interval, and thus would have suppressed N3's transmission.

The implicitly imposed non-transmitting green and orange parts in the *Imin* intervals of N2 and N3 allow for perfectly synchronised equal transmission periods that avoid the short-listen problem and, more importantly, give the same transmission probability to N2 and N3. Thus, even if the suppression mechanism prevents N3 from transmitting in this interval, N3 will get approximately the same chance as N2 and S1 to transmit in the following interval. Note that if a k bigger than one is used, the orange part of the interval is not guaranteed. However, this does not harm New-Trickle, as the beginnings of the transmission periods remain synchronised, which prevents the short-listen problem. In addition, all the neighbours are given the same chance to transmit as early as they can.



(a) (b)

Figure 6 Non-synchronised *Imin* intervals

Other cases of non-synchronised Imin-intervals between neighbours have been experimentally observed, some of which are depicted in Figure 7. For instance, in the cases depicted in Figure 7, whichever node of the two designated by a circle transmit, makes the *Imin* interval of the node designated by a star non-synchronised with the other node designated by a circle. The difference between these cases resides in the amount of shift (non-synchronisation) between the Imin-intervals of these (group of) nodes.

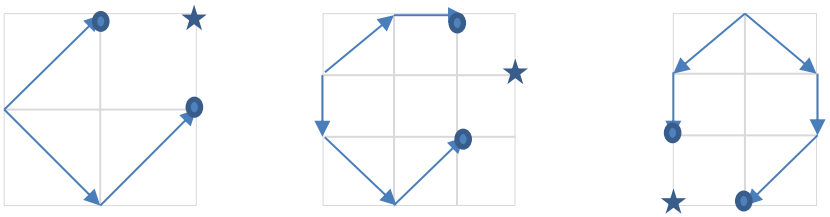


Figure 7 Observed non-synchronised *Imin* intervals. Corner nodes have 3 neighbours, border –non-corner– nodes have 5 neighbours and the others have 8 neighbours per node.

Note that the observed non-synchronised Imin intervals in multi-hop physically lossless networks (note that losses can emerge because of network dynamics) occur between two (groups of) nodes. Nevertheless, for illustrative purposes, this case is generalized in Figure 8 to show that even if more than two neighbours have non-synchronised Imin intervals, the short-listen problem is not observed. Thus, Figure 8 depicts a case of 4 non-synchronised neighbours N1, N2 N3 and N4 which are competing to transmit an update using both New-Trickle (Figure 8 (a)) and Trickle (Figure 8 (b)), although it is rarely the case that the two algorithms can arrive at a similar situation.

Similarly to the discussion carried out regarding Figure 6, no node can transmit in the green part of the intervals; since such transmissions would cause other nodes to start new intervals (which is not the case in this example). Also, it is impossible for nodes N1, N2 and N3 to transmit in the orange part of their intervals when using a redundancy constant $k = 1$, as N1 would have transmitted by the end of its interval and thereby would have suppressed their transmissions.

This leaves only the white parts of the intervals for potential transmissions. As can be seen from Figure 8 (a), the white parts of the intervals are fully synchronised between all the neighbours, giving them an equal chance to transmit. On the other hand, and despite the fact that this example might have favoured Trickle, Figure 8 (b) shows that two of the four neighbours are prevented from transmitting by the listen-only period. The two remaining nodes do not have the same chance to transmit.

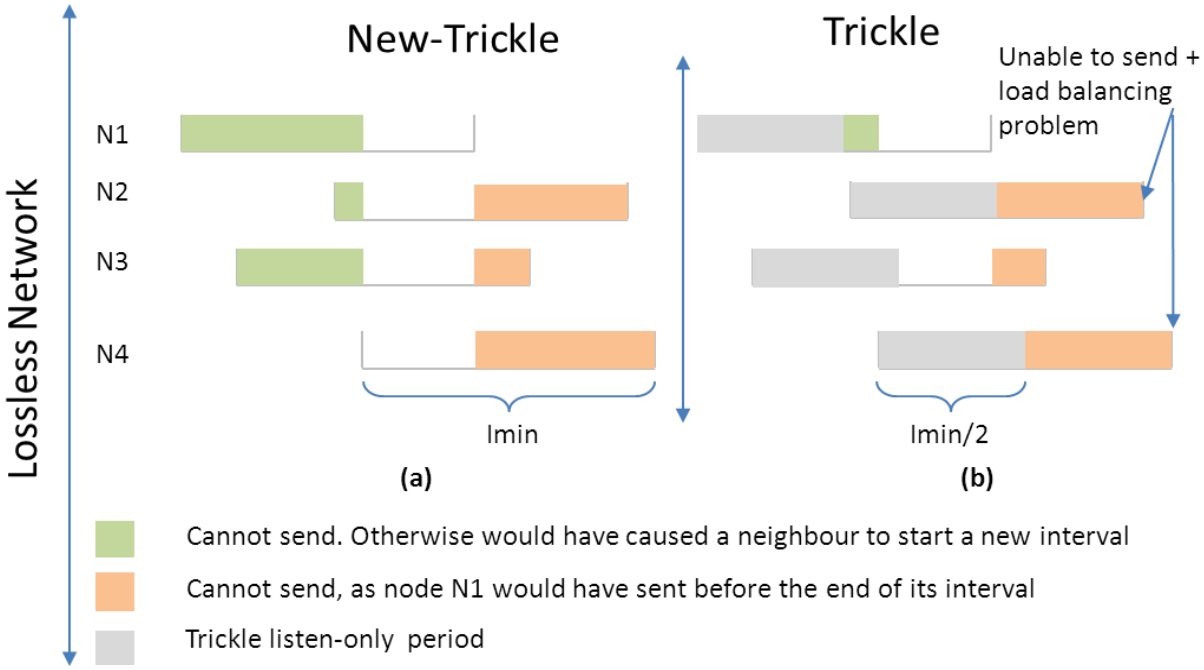


Figure 8 Generalized non-synchronised *Imin* intervals in multi-hop lossless networks (k=1).

4.3. Single-hop, lossy networks

In this subsection, we consider lossy single-hop networks. Losses are very frequent in LLNs (they even form part of their name: low-power and lossy networks). Hence, data drops can be caused by lossy physical links (e.g. as a result of signal attenuation, interferences with other co-existing technologies, propagation patterns) or network dynamics, leading to received corrupted packets. Thus, even if the network can be physically lossless (perfect links), losses can occur as a result of contentions and collisions between concurrent transmissions. This is especially the case in CSMA[4]-based medium access strategies deployed by many LLN technologies (e.g. IEEE 802.15.4 [14]), which suffer from the hidden terminal problem. Note that the authors of [3] have shown that losses can cause Trickle to scale logarithmically with network density, which is also the scalability aimed at by New-Trickle.

Considering losses in the network presented in Figure 6 (a), when a seed node S0 propagates an update, some of its neighbours will hear it, and others will miss it. The nodes hearing it (S1 being one of them) will immediately shrink their intervals and hence can choose t from [0; Imin) without experiencing a short-listen problem. Suppose now that S1 retransmits the update first. Again, some nodes will hear S1's transmission, and others will miss it. Let's analyse these cases:

1. Nodes hearing S1's transmissions can be divided into two categories:
    a. Nodes that have already heard S0's transmission, which simply suppress their retransmissions.
    b. Nodes that have not heard S0's transmission, which immediately shrink their intervals to $Imin$ and can choose t from $[0, Imin)$ without experiencing a short-listen problem with each other.
2. Nodes not hearing S1's transmission can also be divided into two categories:
    a. Nodes that have not also heard S0's transmission; do nothing.
    b. Nodes that have heard S0's transmission, started $Imin$-sized intervals in the past and are still competing to transmit the update.

[4] CSMA: Carrier Sense Multiple Access

Clearly nodes in categories 1.b. and 2.b. are competing to transmit the update and are not synchronised. Thankfully, the short-listen problem is not experienced between these categories. To simplify, take a node from category 1.b. and another from category 2.b., whichever transmits first, annuls the other's transmission, unless the transmission is lost (nothing to do about it).

To generalize this case, let's suppose a single-hop network in which M non-synchronised (group of) nodes are competing to transmit a previously received update. The remaining nodes in the network are denoted by R. We examine in the following points what happens, in the M and R sets, when a first node N1 from M transmits.

1. Suppose that H nodes from M will hear N1's transmission, hence they suppress their transmissions.
2. The remaining $M - H - 1$ nodes from M will miss it; hence they continue competing to propagate the update as they have missed it due to the lossy nature of the network.
3. Now consider that L nodes from R have heard the update for the first time. They start new $Imin$-sized intervals and they will be competing with the $M - H - 1$ nodes to propagate it.
4. The remaining $R - L$ nodes, which either did not hear N1's transmission because of the lossy nature of the network or they are already aware of the update, keep quiet.

The only possibility for short-listen to occur is in step 3 of the above process. We examine this case with the example depicted in Figure 9, where $M = 4$, $L = 0$ and the remaining $R - L$ nodes are already aware of the update and hence are not shown. Figure 9 discusses all the possible transmission combinations of the four nodes (N1, N2, N3 and N4) competing to transmit the update using both New-Trickle and Trickle. Similarly to the analysis conducted in lossless networks, and with the help of colourful keys, Figure 9 (a) shows that the short-listen problem cannot be experienced between these nodes when using New-Trickle. Moreover, the transmission periods of the $Imin$ intervals are fully synchronised between the nodes, giving them an equal chance to transmit even in lossy networks. On the other hand, by deploying the listen-only period in the $Imin$ interval, Trickle avoids the short listen problem at the expense of preventing some nodes from transmitting, along with a load balancing problem showed in Figure 9 (b).

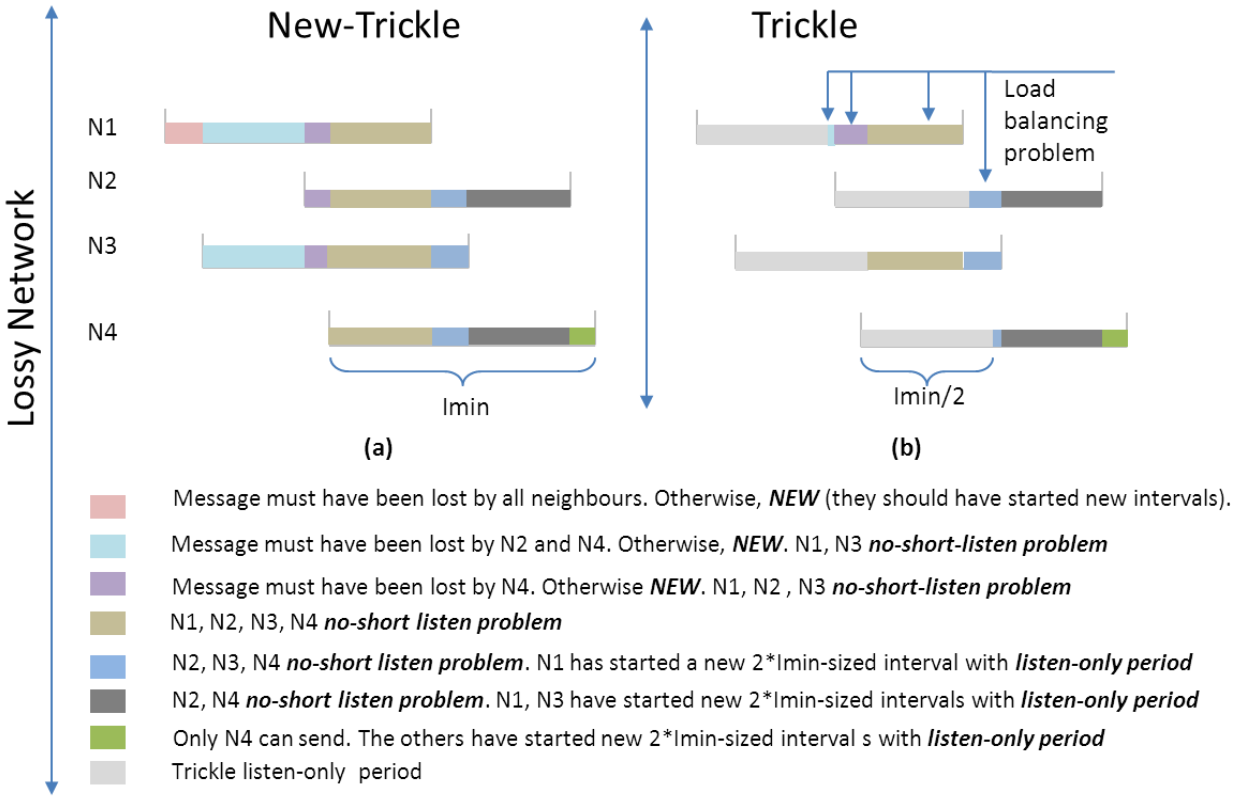


Figure 9 non-synchronisation in lossy networks

4.4. Multi-hop, Lossy Networks

In this section, we discuss the most generic case of multi-hop lossy networks. Various cases of non-synchronised Imin intervals of neighbours competing to propagate an update can occur as a result of overlapping neighbouring regions (Figure 8) or losses (Figure 9) or a combination of both. This scenario can be generalized to the cases discussed in sections 4.2 and 4.3. For instance, if one supposes that losses do not occur in the Imin-sized intervals, then such a case is encapsulated in the generalised scenario depicted in Figure 8. Otherwise, the interactions between neighbours can be captured by the generic case of losses illustrated in Figure 9. Fortunately, in both cases, New-Trickle does not only avoid the short-listen problem but also ensures a perfect synchronisation in the transmission periods of the Imin-sized intervals.

4.5. The Big Picture

Having shown that New-Trickle does not suffer from the short-listen problem in Imin-sized intervals, we put these intervals in the larger context of Trickle's behaviour and determine whether New-Trickle preserves the scalability and robustness of Trickle.

We start with the example of a perfect lossless single-hop network, depicted in Figure 5 (section 4.1). This example shows that Trickle deliberately prevents the originator node N1 from transmitting in the second interval (I_1 interval in Figure 5), as the transmission of N2, N3 or N4 in the Imin-interval forcibly coincides with the listen-only period of the second interval of N1. However, New-Trickle does not guarantee that N2, N3 or N4 transmission in the Imin-interval falls in the second interval of the originator node and hence such a transmission might experience a short-listen problem with I_1's interval of N1. Nevertheless, while New-Trickle allows all the nodes to transmit in the second interval, instead of only N2, N3 or N4 in the case of Trickle, the number of transmissions in the second interval is k for both algorithms. Fortunately, in this lossless perfect scenario, both algorithms generate the same cost, with the advantage of equitable transmission chances provided by New-Trickle.

Let's now take the generic case of lossy networks illustrated through the example depicted in Figure 10. In this case, because of losses, only N2 and N3 hear N1's update. N4 will receive the update from the second transmission (i.e. N2's transmission). The dashed blue lines in Figure 10 show the transmit-listen interplay between Imin-interval transmissions and following intervals' listen-only periods. Although the Imin-sized intervals' transmissions does not experience short-listen with each other for both Trickle and New-Trickle, Trickle makes sure that such transmissions coincide with other intervals' listen-only periods (e.g. N1 and N2 in Figure 10), hence it might help to delete their transmissions. New-Trickle, however; does not provide such a guarantee. This can make New-Trickle transmit more messages in the second interval compared to Trickle. As the second interval deploys the listen-only period, and as this additional cost is caused by losses, the number of transmissions in this interval scales logarithmically with network density, hence preserving Trickle's scalability. Additionally, Trickle's transmit-listen interplay between the second interval transmissions and the third interval listen-only periods decreases. Thereby the additional cost which might be generated by New-Trickle in the third interval is much lower than that in the second interval. This continues so, that from the third interval, the two protocols can generate the same cost. It should be noted that, while we discussed here the case of losses, the small additional cost is mainly caused by non-synchronised $Imin$ intervals and, as shown in section 4.2, non-synchronised $Imin$ intervals can also emerge in lossless multi-hop networks. In addition, it might be expected that the extra cost can slightly increase with increasing k. However, as this cost does not depend on network density, it does not influence the logarithmic scalability of Trickle. Therefore, New-Trickle preserves Trickle's scalability.

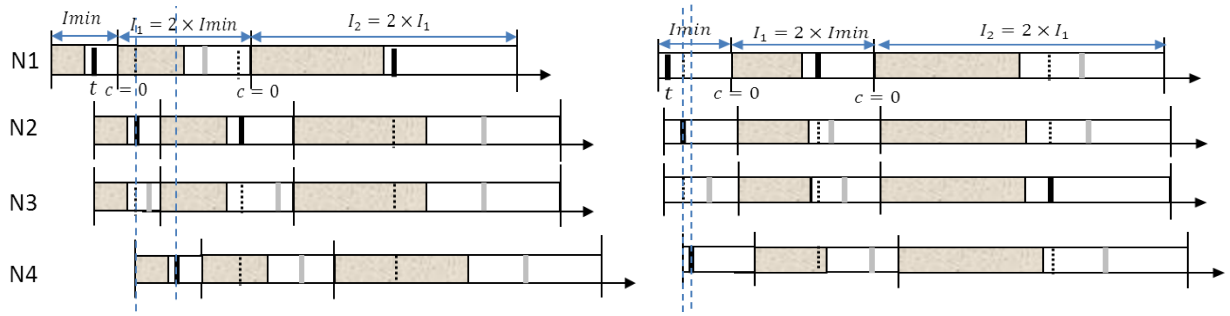


Figure 10 Trickle (left) and New-Trickle (right). The grey rectangle represents the listen-only period. Black lines are transmissions, grey ones are suppressed transmissions and dotted lines are receptions. The dashed blue lines show the transmit-listen interplay between Imin-interval transmissions and next intervals listen-only periods.

It should be noted that some of this small additional cost that only affects a few following intervals can be mitigated. Thus, by allowing a node which transmitted during an Imin interval to start listening immediately, most of the second interval's additional cost can be suppressed. However, it might be better to not delete such a cost, as it can help in uniformly distributing the transmission load, as will be shown in the following section.

Having demonstrated that the proposed optimisation does not influence Trickle's scalability, we present in the following section some further expected benefits of the New-Trickle algorithm.

5. Expected Other Benefits of New-Trickle

In previous sections, we showed that choosing t from $[0; \text{Imin})$ can allow New-Trickle to propagate dramatically faster without resulting in a short-listen problem between competing neighbours. We also demonstrated that although a small additional cost can occur in New-Trickle, this cost might only be observed in a few (e.g. 2-3) intervals following $Imin$, and that it does not influence Trickle's scalability. In this section, we outline some other benefits that can result from New-Trickle, which address the remaining criticisms discussed in section 2.2.

5.1. Load Balancing

Trickle inherits a balanced load distribution arising from the uniform random choice of transmission time. However, this balanced load can be challenged by the listen-only period as explained in section 2.2. As shown in that section, unbalanced load distribution has more chances to occur in small intervals (especially Imin-sized intervals), where it has the most impact. Additionally, it was shown in section 2.2 that the listen-only period of Imin-intervals may explicitly stop some transmissions, thus preventing parts of the network from being quickly updated. Throughout the above analysis (section 4), we showed that New-Trickle gives all competing nodes similar chances to transmit an update, which allows it to solve this serious issue. In what follows, we focus on how New-Trickle helps to bring a balanced load distribution. To this end, we use the generic case of lossy networks depicted in Figure 10.

As can be seen from Figure 10, Trickle imposes on every node to wait for at least the size of the listen-only period before propagating an update. This shifts the intervals of the receivers by at least $\text{Imin} / 2$ from the originator. A receiver form those (e.g. node N2 in Figure 10) has to wait for at least another $\text{Imin} / 2$ before transmitting. As a result, a receiver of such an update (for instance, node N4 in Figure 10) is again shifted by at least $\text{Imin} / 2$ from N2 and by Imin from the seed. This process gets aggravated under heavy losses, which adds to the amount of non-synchronisation between the intervals of neighbouring nodes. This in turn might give some nodes more chances to transmit in the following intervals as discussed in section 2.2. New-Trickle, however, does not impose any restriction on nodes competing to transmit an update. Hence, in addition to giving competing nodes the same chances to transmit, it allows for a smaller shift in the intervals of neighbouring nodes, as shown in Figure 10. This helps to fairly redistribute the cost between neighbours as early as the first interval. Note that this discussion can open doors for other optimisations to try to resynchronise node intervals and hence might remove the need for a listen-only period.

5.2. Less Inconsistent Transmissions

New-Trickle showed a perfect synchronisation in the transmission periods of the Imin-intervals of neighbours competing to propagate an update (Figure 8 and Figure 9). Therefore, New-Trickle can theoretically ensure the expected number of k transmission per an $Imin$ interval in lossless networks ($k \times log\ (n)$ in lossy networks) when $k = 1$. For the other values of k, there might be a higher probability to approach the theoretical limit. On the other hand, a half-interval listen-only period bounds the number of transmissions generated in an interval by a constant, asymptotically approaching $2 \times k$ in a lossless network and $2 \times k \times log\ (n)$ when considering losses. Therefore, the number of transmissions in an Imin-interval is fundamentally smaller in New-Trickle than in Trickle. This is important not only in minimising the number of transmissions in the $Imin$ interval, but it is also important in reducing chances for contentions, collisions and hidden terminals in this critical interval where most of Trickle's propagations are done.

However, as discussed in section 2.2 and demonstrated in the previous section, the listen-only period of the $Imin$-interval in Trickle can prevent a node from transmitting in this interval, especially when using a redundancy constant of one ($k = 1$). This fact might allow Trickle to generate fewer messages in an $Imin$ interval for such a value of k in some scenarios.

5.3. More Room for Contentions

Doubling the size of the potential transmission part of an $Imin$ interval can give nodes more time for contentions, which helps to minimise the probability of collisions and hidden terminals. Therefore, New-Trickle might improve the performance of the suppression mechanism. This is especially important when opting for very small $Imin$ intervals or/and deploying New-Trickle in dense networks. However, since New-Trickle ensures similar chances for nodes to transmit in an $Imin$-Interval, whilst Trickle explicitly prevents some nodes from transmitting, Trickle can still, in some cases, generate less overhead in an $Imin$ interval.

Having introduced New-Trickle, analysed its scalability and trade-offs, and demonstrated its benefits, we present in the following section the evaluation methodology and design followed to assess its performance.

6. Evaluation Methodology

In this section, we outline the evaluation methodology and experimental design of this study. We conduct realistic simulations and public testbed experiments in order to evaluate the performance of New-Trickle. Simulation experiments give us controlled environments, while public testbed experiments validate simulation results in real-world deployments. To put results into context, we compared New-Trickle with Trickle and Short-Trickle; a version of Trickle without the listen-only period. In what follows, we describe the implementation, the performance metrics and the experimental design.

We used the Contiki OS Trickle library[5] as a basis for our modifications and evaluations. We used the default setup of the library, which takes care of basic validity checks and adjustments, along with compensating for clock drifts. We also implemented our modifications in TinyOS for the sake of validating the results in another environment.

We setup two Trickle scenarios

1. **Setup 1:** In this setup, we generate just one inconsistent transmission, which gets propagated in the network. Thus, a seed node in the upper-left corner of the network issues a new packet (identified by a sequence number) once. The network uses New-Trickle, Trickle and Short-Trickle to propagate the packet and keep gossiping about it until the end of the simulation time. This setup is used to get a clear understanding of New-Trickle.
2. **Setup 2:** Inspired by the Contiki OS Trickle example, we created an abstract Trickle-based application in which a seed node periodically injects new packets (identified by new sequence numbers) in the network. In this abstract application:
    - Receiving a packet with the same sequence number implies a consistency
    - Receiving a new packet (greater sequence number than receiver's version) implies an inconsistency for which the receiver updates its data and contend to propagate the update.
    - Receiving an old packet (smaller sequence number than receiver's version) implies also an inconsistency in this abstract application. In this case, the receiver shrinks its interval to $Imin$ and contends to transmit its data in order to bring the outdated neighbour up to date.

    Note that by periodically injecting new Trickle messages in the network, we deliberately create a network dominated by inconsistent traffic in order to clearly show the impact of the transmit-listen interplay on New-Trickle's performance.

Our Trickle-based applications described in Setup 1 and Setup 2 is developed over UDP (User Datagram Protocol) using Contiki's micro IPv6 network stack (uIPv6) at the network layer and a CSMA-based protocol at the MAC (Media Access Control) Layer. At the RDC (Radio Duty Cycling) layer, we opted for a non-duty cycled network. Using a non-duty cycled network allows us to focus on New-Trickle's performance rather than the effects of duty cycling.

The network configuration used in all our simulations is expressed in the following points:

- Nodes booted-up randomly in a 10-second period in order to avoid initially synchronised node clocks.
- The MAC layer retransmissions were disabled for the sake of avoiding any interplay between MAC and Trickle retransmissions. Under this configuration, if a Trickle transmission undergoes collisions it will be lost. Trickle

[5] https://github.com/contiki-os/contiki/blob/master/core/lib/trickle-timer.c

takes care of retransmissions, since other nodes will transmit if they do not hear anything. In a worst case, out-dated nodes transmissions will create inconsistencies triggering retransmission of updates. Note that disabling the MAC layer retransmissions allows us also to correctly measure the number of Trickle's transmissions.

- We used a non-duty cycled network (always on radios) in order to avoid the effects of RDC mechanisms (e.g. delays, multiple radio transmissions…etc.) on the evaluated protocols. Note that RDC effects are being separately examined by the authors and will be reported in a future work.

In all our experiments, we focused on three main performance metrics defined below:

- ***Transmissions/Interval:*** this metric is measured as the ratio between the total number of all transmissions generated by an evaluated protocol and the number of intervals ran by the evaluated protocol during the simulation time. Note that as Trickle Intervals have different sizes and a protocol might have slightly less or more intervals than another, we normalized the number of intervals across the evaluated protocols in other to provide fair comparisons. Note also that we sometimes (especially when using Setup 2) report the total number of transmissions during the simulation time.

- ***The consistency time:*** the consistency time is the time it takes for the whole network to be aware of an update from its first appearance in the network. It is measured as the difference between the time when the last node gets updated and the time when the update first appeared. It shows how fast a protocol can resolve inconsistencies.

- ***The number of inconsistent packets:*** this metric measures the number of inconsistent packets generated by each protocol, i.e. the number of transmissions in an Imin interval. As we only modify Trickle's behaviour in an Imin-sized interval, this metric is important to show how New-Trickle impacts the number of packets generated in such an interval. In addition, it allows validating the discussion carried out in sections 4 and 5.2.

We also measured other secondary metrics in order to help explain the results, such as the number of packets generated in the second, third and remaining intervals.

Varied Parameters

- ***The minimum interval size:*** the minimum interval size Imin is varied from 62 milliseconds (ms) to 20 seconds in order to accommodate various Trickle use-cases, ranging from those deploying very small Imin values, such as RPL (recommended 8 ms) and CTP (default 64 ms), to those adopting large Imin values, such as in distributed service discovery protocols [8]. Note that while the RPL specification suggests an Imin value of 8 ms, the Contiki RPL implementation, for instance, defaults it to 4 seconds.

- ***Network density:*** Since network density is the main factor dictating Trickle's scalability, it was paid special attention in our evaluations. Thus, the number of nodes was varied between 16 and 400, thereby internetworking up to 400 nodes simultaneously in the case of single-hop networks. In multi-hop networks, we varied the number of nodes in a given area between 36 (allowing us to test a sparse network of an average density of 4 neighbours/node) and 196 nodes, which gives a 36 neighbours/node average density. We also fixed the number of nodes at 400 and varied the side of the square deployment area between 300 and 400 meters, thus varying the average density between 36 and 4 neighbours/node respectively.

- ***The redundancy constant k:*** The redundancy constant k was varied between 1 and 9 for the sake of accommodating the needs of various Trickle-based deployments such as MPL ($k = 1$), RPL ($k = 10$) or CTP with an infinite redundancy constant (achieved through $k = 0$).

- ***Physical success rate:*** to see the impact of loss on New-Trickle, we varied the reception success ratio of a packet in the cooja simulator (SUCCESS_RATIO_RX) between 10% (giving a 90% physical loss rate) and 100% (lossless networks). We configured zero retransmissions at the MAC layer, in order to ensure that each Trickle transmission results in exactly one MAC transmission that undergoes the reception success probability. Note that cooja does not simply generate a uniform random reception success probability based on the configured success ratio. Instead, it introduces some realism to decide which node undergoes the SUCCESS_RATIO_RX based on the ratio between reception and transmission powers. Since the Unit Disk Graph Medium (UDGM) only models attenuations as a function of distance, and as the reception power can be proportional to the square distance

between the sender and the receiver, cooja calculates, for each node, the $Reception\ success\ probability$ as follows[6]:

$$Reception\ success\ probability\ =\ 1.0\ -\ ratio * (1.0 - SUCCESS\_RATIO\_RX)$$

Where

$$ratio\ =\ distanceSquared\ /\ distanceMaxSquared$$

distanceMaxSquared is the square of the communication range and distanceSquared represents the square distance between a receiver-sender pair.

- ***Transmission power:*** In testbed experiments, we varied the maximum transmission power of a sender over 5 transmission levels; namely, levels 11, 15, 18, 23 and 31, representing transmission powers at -10, -7, -5, -3 and 0 dBm respectively.
- ***Network Topology:*** We used both grid and uniformly distributed topologies along with an irregular testbed deployment. In both deployments, we evaluated sparse and dense scenarios.

Note that we also combined the above parameters. For example, we evaluated New-Trickle in a very lossy, dense, uniformly deployed network with a big value of k. Finally, it should be noted that other parameters were implicitly varied in our evaluations. For instance, the network diameter (hop count) is varied by testing 100- and 400-node networks in similar deployments. Results from the 100-node network are depicted in Appendix C.

Table 1 main evaluation parameters

Parameter	Value
Radio model	UDGM (range = 50 m) / 500 m in single hop networks
Default deployment area	300 m x 300 m
MAC and RDC	CSMA with 100% duty cycle
Application	Trickle/UDP/6LoWPAN
Message Payload	20 Bytes
MAC retransmissions	0 (Trickle takes care of retransmissions)

7. Results and Discussions

The main simulation results discussed in this section are from evaluating setup 1 in a dense reference scenario containing 400-nodes deployed in a 20x20 grid, which allows for a 36 neighbour/node density in multi-hop networks. The network diameter was about 13 hops. When necessary, results from other deployments, especially sparse networks, are discussed. The rest of the obtained results are reported in the appendices. To this end, we start by discussing results from multi-hop networks, and then focus on single-hop lossy networks. Finally, we discuss the results obtained from the Indriya testbed [15]. Unless otherwise stated, the default value of Imin is one second and that of k is one in all our simulations. Each simulation runs for 10 virtual minutes and is repeated 25 times. We plot in the following graphs the mean with its standard error.

7.1. Simulated Multi-hop Networks

In the following subsections we discuss the results obtained from running the evaluated protocols in the reference scenario. We first discuss the main metrics and then quantify the small additional cost generated by New-Trickle, when varying network density, loss rate, $Imin$ and k. More results from dense uniformly distributed networks are presented in Appendix B.

7.1.1. Main results

Figure 11 presents results obtained from the reference multi-hop scenario. It contains six subfigures, each presenting the consistency time, transmissions/interval, and the number of inconsistent transmissions registered by the evaluated protocols when varying the parameters described above.

Figure 11 (a) and (b) show the performance of New-Trickle, Trickle and Short-Trickle under different network densities. As can be seen from Figure 11 (a), New-Trickle propagated more than two times faster than Trickle when varying the deployment area of a 400-node network. The biggest difference was observed in a sparse network of 4

[6] https://github.com/contiki-os/contiki/blob/master/tools/cooja/java/org/contikios/cooja/radiomediums/UDGM.java

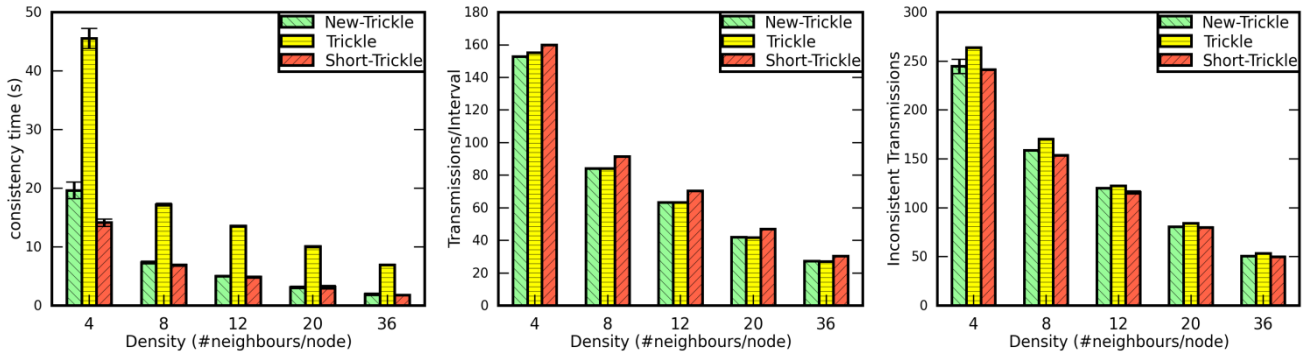
neighbours/node density. Such a performance is achieved at approximately the same cost as Trickle, and even lower in sparse networks. This can be explained by the fact that Trickle prevents some nodes from transmitting in the $Imin$ interval, which might require more transmissions in order to achieve consistency in sparse networks. This is confirmed by the number of transmissions generated in the $Imin$ interval depicted in the third column of Figure 11 (a). This graph shows that New-Trickle generated fewer inconsistent packets than Trickle with the biggest gap observed in the sparse network of 4 neighbours/node. This graph validates our assertions about synchronised transmission periods of $Imin$ intervals, allowing New-Trickle to send fewer packets. Compared to Short-Trickle, New-Trickle achieved similar consistency times, with the gap decreasing with increased network density. This is so, since in sparse networks the first propagation wave might get stopped, by the suppression mechanism, before it starts again in the second interval, where New-Trickle deploys the listen-only period. Finally, Short-trickle generated the biggest cost, since it suffers from the short-listen problem. These results are confirmed when varying the number of nodes in the network as shown in Figure 11 (b). For a better understanding of New-Trickle's performance in sparse networks, a full evaluation in a 400-node sparse network under heavily inconsistent traffic (setup 2) is presented in Appendix A.

Concerning the impact of losses, Figure 11 (c) presents the performance of New-Trickle, Trickle and Short-Trickle when varying the physical loss rate. As can be seen form this figure, New-Trickle approached the propagation time of Short-Trickle even in a worst case of 90% loss rate. The gap between the two protocols decreased with increasing success rates. This is explained by the fact that in very lossy networks there are more chances of losing an update for the first time which postpones its delivery until the next interval. In such an interval, New-Trickle deploys the listen-only period, while Short-Trickle does not, allowing it to propagate faster. In all cases, the consistency time of both Short-Trickle and New-Trickle was about four times lower than that of Trickle. While Short-Trickle achieved such a performance by generating more packets, New-Trickle generated approximatively the same cost as Trickle. Upon a closer look, New-Trickle generated slightly more packets than Trickle which are more visible in lossy networks. This is explained by the transmit-listen interplay benefiting Trickle, which is quantified in the following subsection. However, the packets generated by New-Trickle in the $Imin$ interval are still fewer than that of Trickle, since the implicit synchronisation discussed in section 45.2 is loss independent.

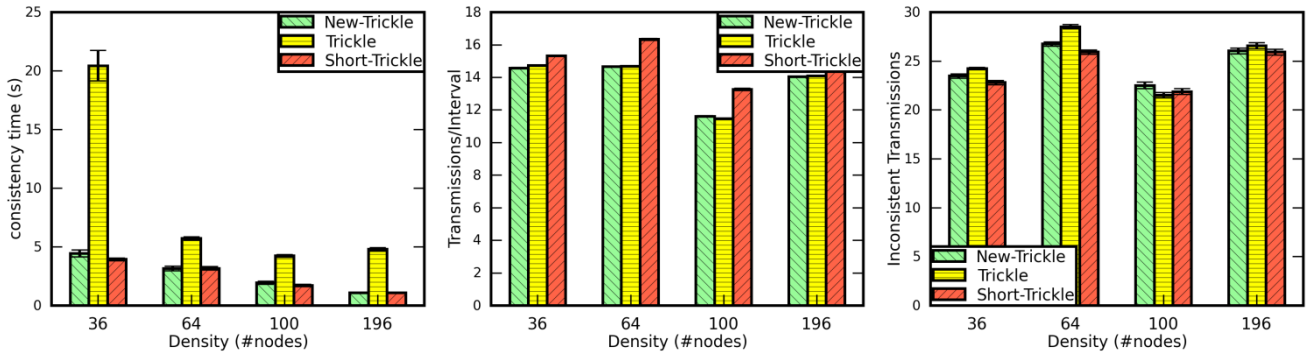
Figure 11 (d) depicts the performance of New-Trickle when varying $Imin$ in a physically lossless network. As expected, New-Trickle achieved network consistency as quickly as Short-Trickle. Interestingly, the propagation time of New-Trickle does not heavily depend on $Imin$ as is the case for Trickle, thereby allowing New-Trickle to propagate new updates seven times faster in an $Imin$ of two seconds. This gap is expected to increase with increased $Imin$ values. While short-Trickle achieved such a propagation speed generating more messages, the cost of New-Trickle is similar to that of Trickle. It should be noted, however, that a small difference in the cost of the two protocols can be observed, and it is more visible for smaller $Imin$ values. This can be due to two main reasons; namely, losses and unbalanced load distribution, both benefiting Trickle in dense networks. Thus, even though the network is physically lossless, losses can always occur because of collisions and hidden terminals, which are more likely to occur in small contending periods i.e. smaller $Imin$ intervals. On the other hand, since Trickle explicitly prevents some nodes from transmitting, it minimises the number of contenders in an $Imin$ interval, which in turn minimises losses. This is confirmed from the inconsistent transmissions graph, which shows that New-Trickle generated about 60% more transmissions in an $Imin$ interval of 62 ms. This gap decreased dramatically, such that in an $Imin$ of 125 ms, New-Trickle only added about 20% extra packets, and from an $Imin$ of one second, New-Trickle generated less overhead than Trickle as its inherent synchronisation outperformed collision effects. Note that this only benefits Trickle in dense networks, while it constitutes a drawback for Trickle in sparse networks, as shown in Appendix A. Thus, in sparse networks, New-Trickle outperformed Trickle in both time/cost aspects when varying $Imin$ using the default value of $(k = 1)$.

Finally, Figure 11 (e) and (f) present the performance of the evaluated algorithms when varying the redundancy constant in both physically lossless and very lossy networks. As can be seen from these figures, increasing k decreased slightly the network consistency time while increased considerably the cost of the three protocols in both lossy and lossless dense networks. This observation remains valid in sparse networks, as can be seen from Appendix A. Concerning individual protocols performance, Figure 11 (e) shows that New-Trickle propagated about 3.5 times faster than Trickle while also generating fewer messages when increasing k. This can be explained by the fact that in physically lossless dense networks, New-Trickle suffers less from the transmit-listen interplay, while at the same time benefiting from its implicit synchronisation in an $Imin$ interval. This is confirmed from the graph of generated inconsistent packets, which shows that the additional cost of Trickle increased with increasing k values. Note that this is expected to be even better in

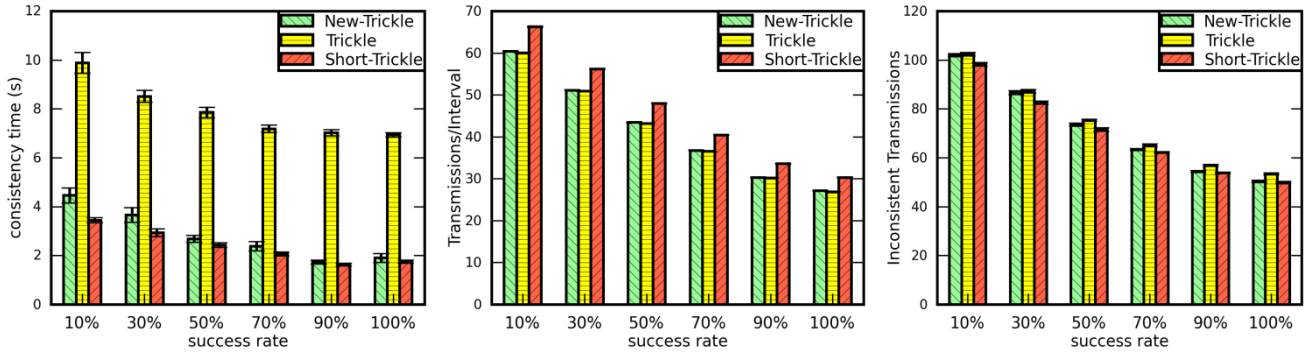
dense networks deploying bigger values of k. However, this is not the case in sparse networks experiencing heavily inconsistent traffic. Thus, in such a configuration, New-Trickle generated slightly more transmissions than Trickle when k increased, as shown in Appendix A. This can be due to the transmit-listen interplay helping Trickle's consistency counter to reach k without actually generating k packets. When taking link drops into consideration, Figure 11 (f) shows that New-Trickle can compensate for the additional cost caused from lacking the transmit-listen interplay by using the transmissions saved in the $Imin$ Interval.



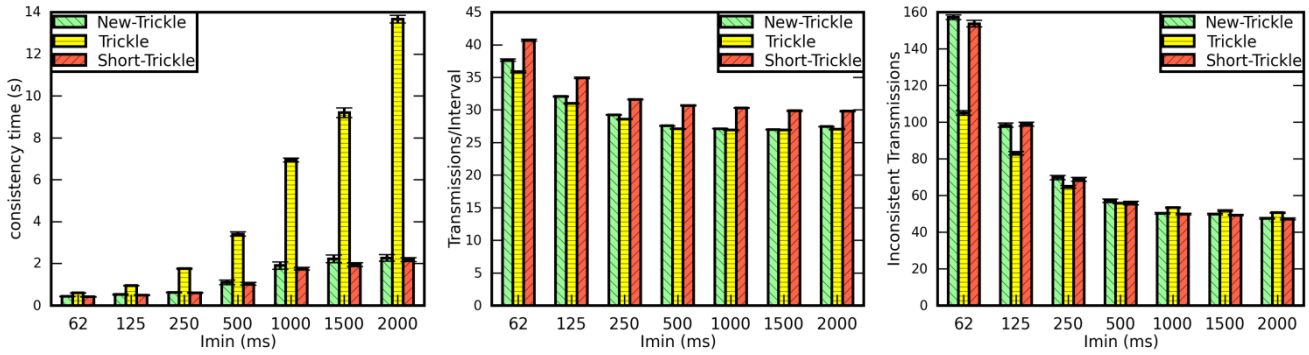
(a) Varying network area, success rate = 100 %



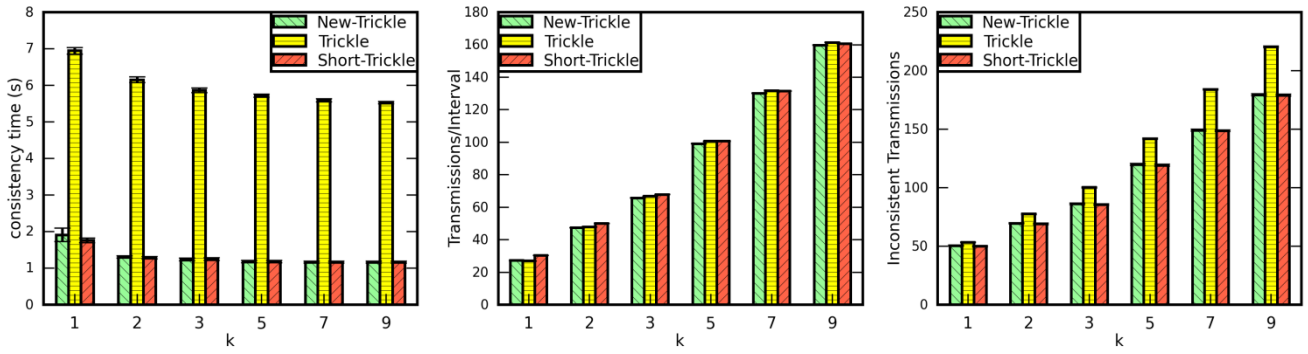
(b) Varying number of nodes, success rate = 100 %



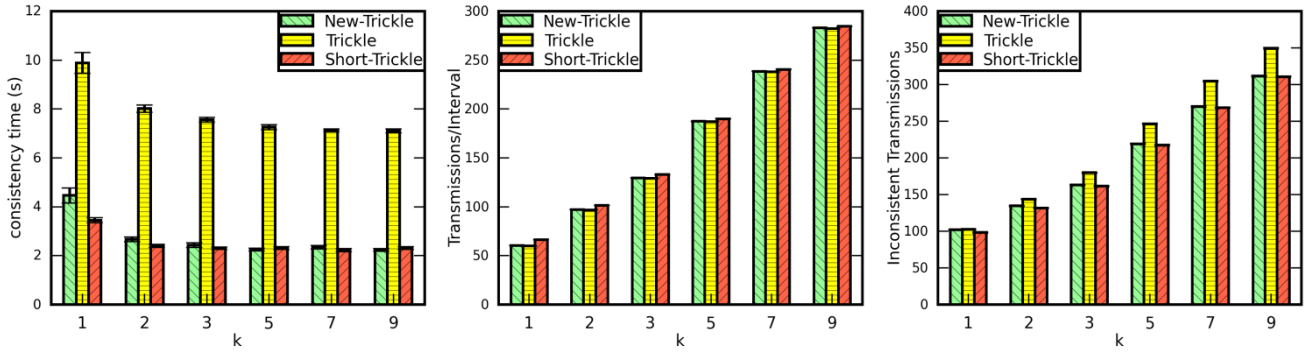
(c) Varying physical success rate



(d) Varying $Imin$, success rate = 100 %



(e) Varying k, success rate = 100%



(f) Varying k, success rate = 10%

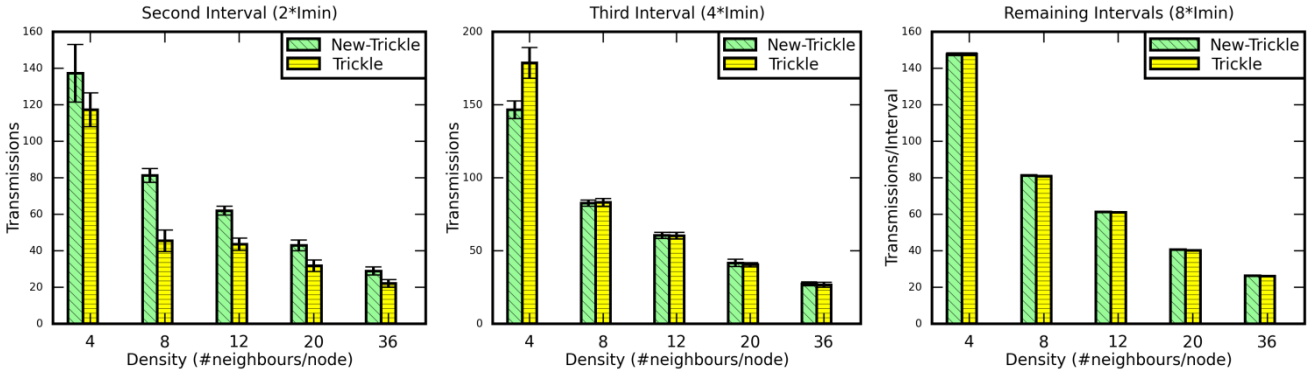
Figure 11 Performance evaluation in multi-hop networks

Having analysed the main performance of New-Trickle in multi-hop networks, we discuss and quantify in the following subsection the additional cost observed in some of the above results and confirm with experimental data that this cost does not affect the logarithmic scalability of New-Trickle under any of the varied parameters.

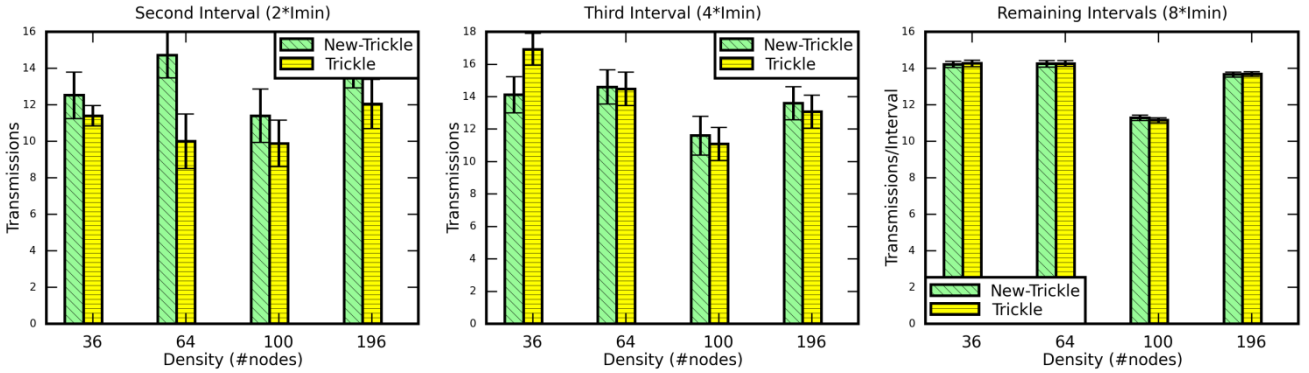
7.1.2. Quantifying the additional cost

This subsection discusses the small additional cost observed in some of the graphs depicted in Figure 11 and demonstrates that New-Trickle's scalability is not impacted by such a cost. To this end, we plot in Figure 12 the number of transmissions generated by Trickle and New-Trickle in the second, third and remaining intervals. Overall, Figure 12 shows that the small additional cost does not violate the logarithmic scalability of New-Trickle and it disappears after a few intervals following *Imin*. Thus, from the third interval such a cost becomes unnoticeable. In the remainder of this section, we discuss the impact of such a cost when increasing the network density, in order to demonstrate that the logarithmic scalability of Trickle is preserved. Then, the discussion is shifted to only interesting cases.

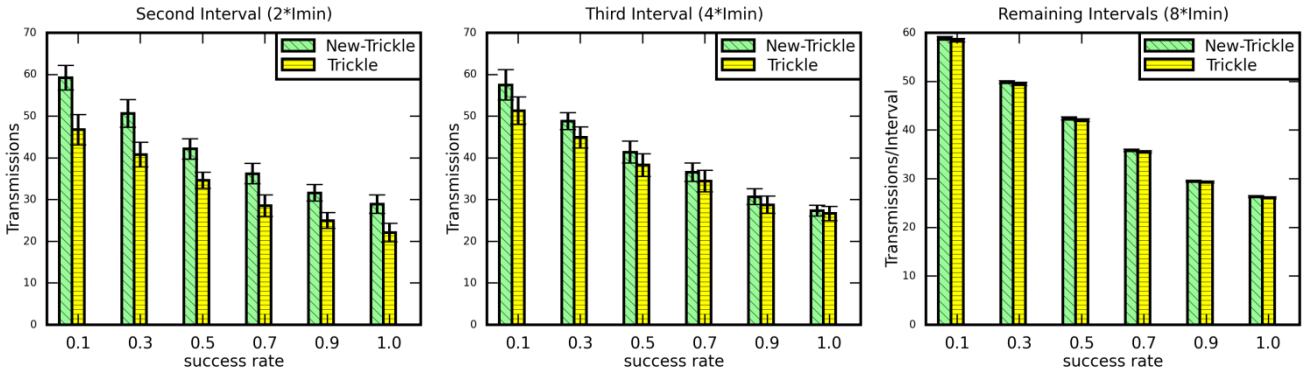
Figure 12 (a) and (b) show that New-Trickle's additional cost is density independent and that it disappears as soon as the third interval (second interval after *Imin*). The main cause of this cost is the transmit-listen interplay ensured by Trickle and not guaranteed by New-Trickle. Such a cost is mainly caused by losses and the multi-hop nature of the network, and it might increase with increasing k. Therefore, when these three parameters are combined, the cost saved by New-Trickle in the *Imin* interval might not be able to compensate for all, and hence a very small additional cost can be observed (Figure 12 (f)). Finally, it is interesting to observe that, while in the second interval Trickle always generated fewer packets than New-Trickle, in the third interval New-Trickle can transmit less. This is especially true in sparse networks (Figure 12 (a) and (b)) and when using smaller values of *Imin* (Figure 12 (d)). When these two parameters are combined, New-Trickle always leads to a fewer cost in such an interval, even under heavy inconsistent traffic, as can be seen from Appendix A.



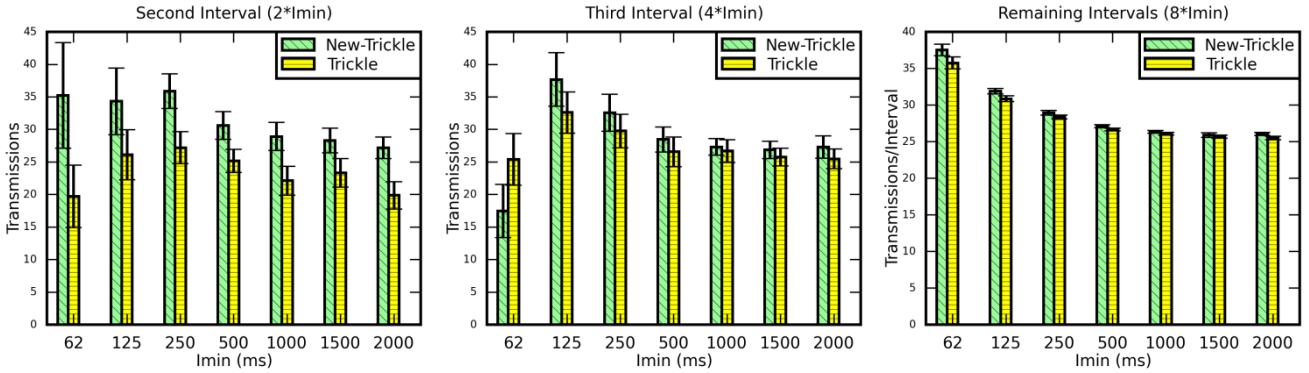
(a) Varying network area, success rate = 100%



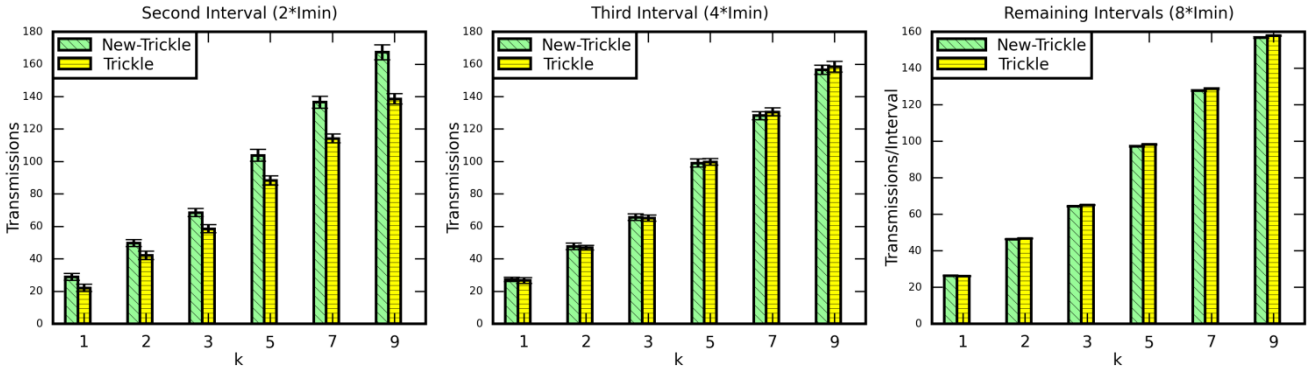
(b) Varying number of nodes, success rate = 100%



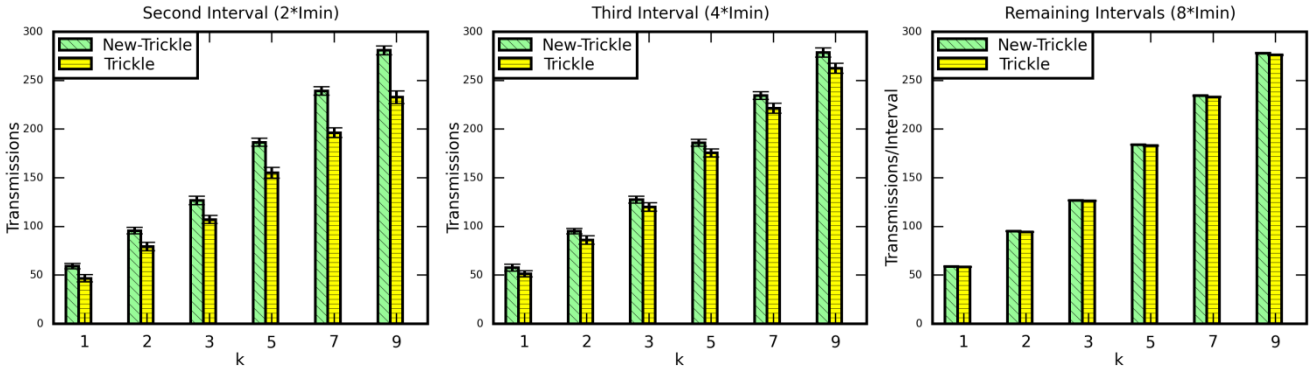
(c) Varying the physical success rate



(d) Varying *Imin*, success rate = 100%



(e) Varying k, success rate = 100%



(f) Varying k, success rate = 10%

Figure 12 quantifying the additional cost in multi-hop networks

7.2. Single-hop networks

In this subsection, we discuss the performance of the three evaluated protocols in the reference single-hop scenario. Since in lossless single-hop networks, a transmission can reach all the nodes, and hence the consistency time of the three protocols would be the same, we present in the following figures results from lossy networks with a 10% configured success ratio. Note that since cooja calculates the reception success probability of a node as a function of its distance to the sender, only the farthest nodes undergo the configured ratio and hence the 90% loss ratio should not be taken literally.

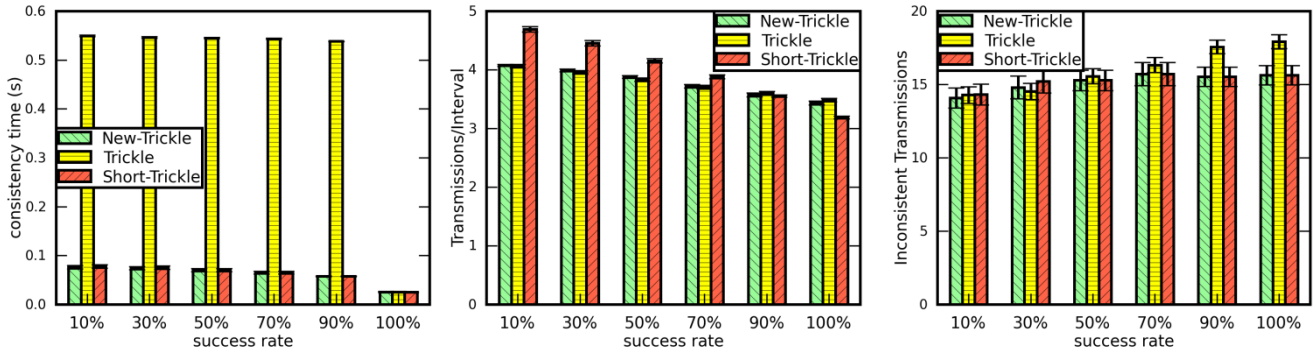
7.2.1. Main results

Figure 13 presents the results obtained from the reference single-hop scenario. As in the previous case, this figure is divided into four subfigures, each depicting the performance of the evaluated protocols when varying physical success rate, network density, Imin and k.

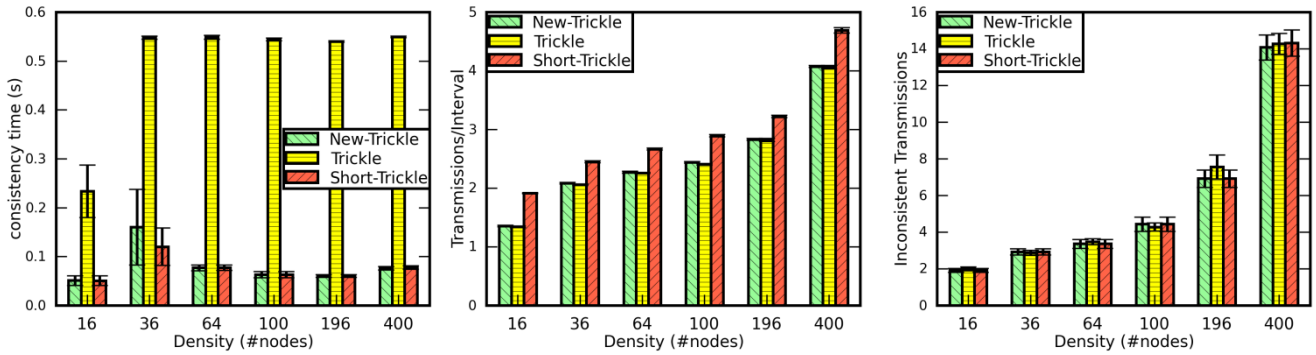
Figure 13 (a) depicts the consistency time and the transmission cost of the three evaluated protocols when varying the reception success rate. New-Trickle propagated about five times faster than Trickle with an approximate similar cost (the small extra cost is discussed in the following subsection). Note that in the particular case of 100% success rate, Trickle achieved the same propagation speed as New-Trickle and Short-Trickle. Even, the latter sent fewer packets than both Trickle and New-Trickle. This is so because the network was implicitly synchronised, since the inconsistent transmission was simultaneously received by all the nodes.

When varying the density (Figure 13 (b)), New-Trickle propagated as fast as Short-Trickle and generated only a very small additional cost compared to Trickle, while propagating more than six times faster. Similar performance was observed when varying Imin. Thus, as can be seen from Figure 13 (c), New-Trickle's propagation time is loosely coupled with the value of Imin allowing it to propagate about 11 times faster than Trickle for an Imin value of two seconds. This is achieved at approximately the same cost as Trickle. On the other hand, Short-Trickle's cost increased drastically with increasing losses. Finally, Figure 13 (d) presents the performance when varying the redundancy constant. As expected, New-Trickle propagated around eight times faster than Trickle and achieved network consistency as quickly as Short-Trickle. The small additional cost incurred by New-Trickle for achieving such a performance is more visible when increasing k. The reasons behind this are discussed in the following subsection. It should be noted that in all cases New-

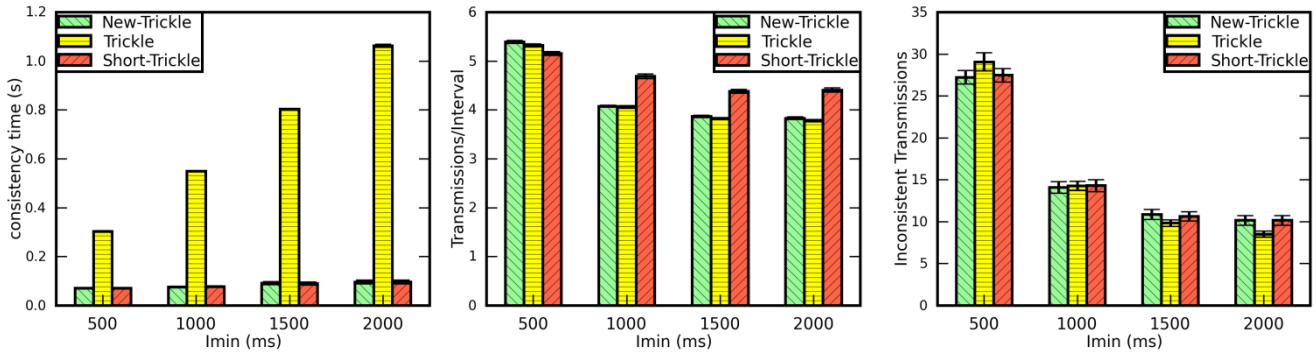
Trickle generated fewer transmissions in the Imin interval, thanks to its inherent synchronised transmission periods in such an interval.



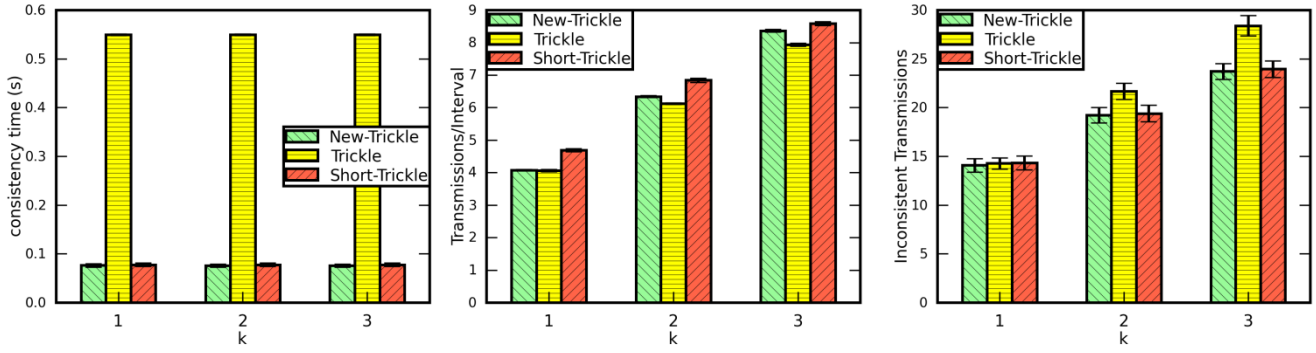
(a) Varying the physical success rate



(b) Varying the number of nodes. success rate = 10%

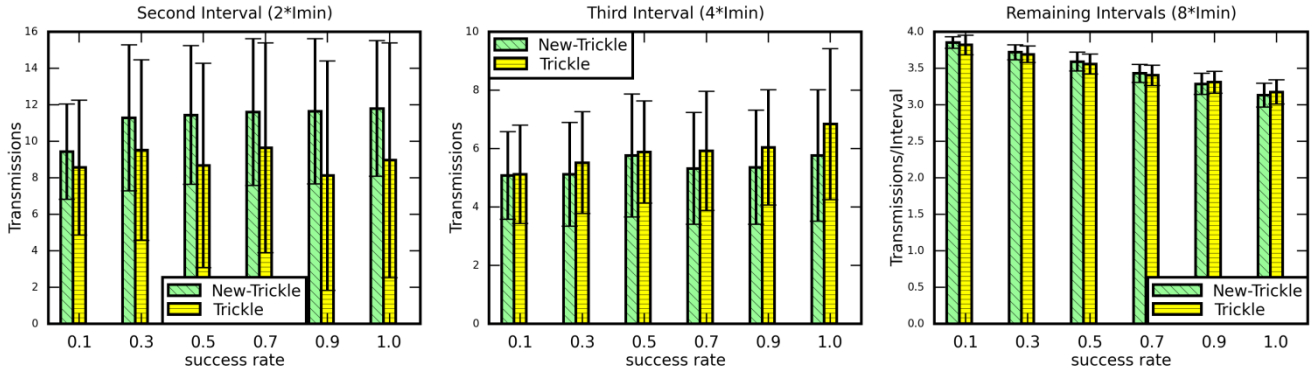


(c) Varying *Imin*, success rate = 10%

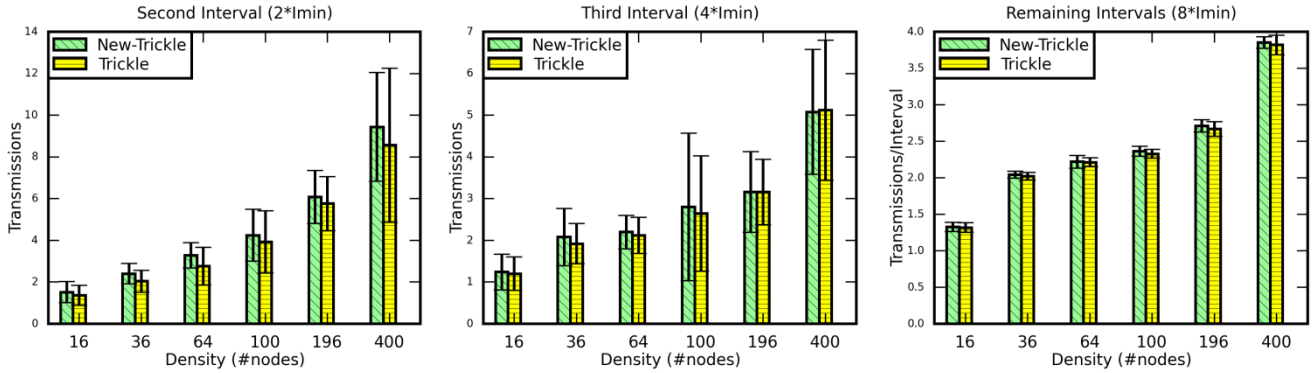


(d) Varying k. success rate = 10%

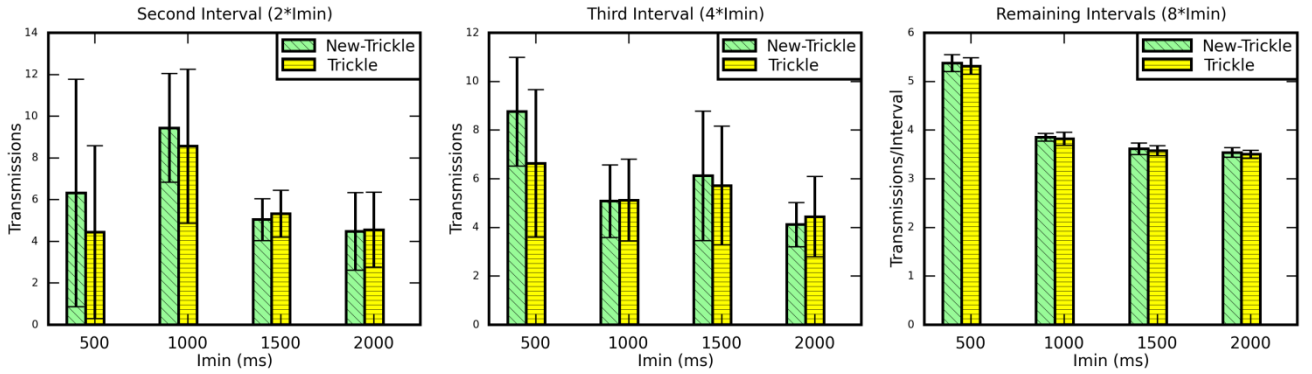
Figure 13 Performance evaluation in single-hop networks



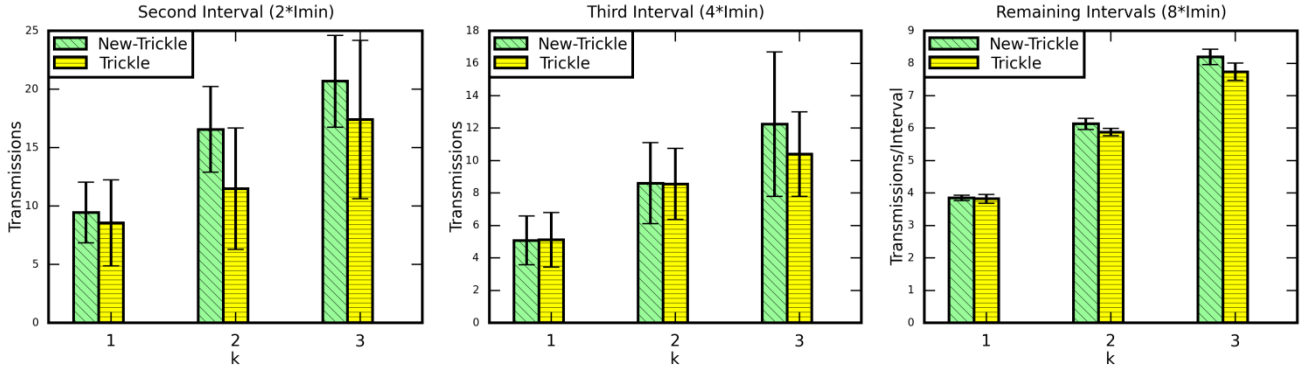
(a) Varying the physical success rate



(b) Varying the number of nodes, success rate = 10%



(c) Varying *Imin*, success rate = 10%



(d) Varying *k*, success rate = 10%

Figure 14 quantifying the additional cost in single-hop networks

7.2.2. Quantifying the additional cost

As in multi-hop networks, we discuss in this subsection the additional cost generated by New-Trickle in single-hop networks and show that it does not impact New-Trickle's logarithmic scalability.

Figure 14 quantifies the additional cost introduced by New-Trickle in the second, third and remaining intervals. Similarly to multi-hop networks, this additional cost does not depend on the number of nodes in the network, as can be seen from Figure 14 (b), and it can disappear after the third interval when using a default value of $k = 1$. Similar trends were also observed when varying the physical success ratio (Figure 14 (a)) and the minimum interval size Imin (Figure 14 (c)).

When k increased however, the additional cost generated by New-Trickle was still observable even in the remaining intervals. This can be explained by the transmit-listen interplay benefiting Trickle in a network of 400 internetworked direct neighbours. Thus, combining a very dense single-hop network, suffering severe losses and using increased k values allowed such an additional cost to be made more visible. Nevertheless, and even in this very constrained scenario, the additional cost generated by New-Trickle remains independent from the network density and therefore does affect New-Trickle's scalability.

7.3. Empirical Study

As mentioned earlier, New-Trickle was also evaluated in the public large-scale Indriya testbed [15]. Indriya contains around 100 active motes irregularly deployed in a three floor building at the national university of Singapore. The layout of the Indriya testbed is presented in Appendix E. At the time of experimentation almost all middle floor nodes were off, leaving us with 65 motes and a good opportunity to test in an irregular, faulty real-world scenario. Using setup 2, the seed (node 21 in the third floor) injected a new packet every 60 seconds. This is so to create a network dominated by inconsistent traffic, in order to show the impact of the observed small additional cost in the simulation results. Each experiment was run for 30 minutes and was repeated three times. The default value of $Imin$ was half a second and that of k was one. As in the simulations, we report in the graphs the mean along with its standard error.

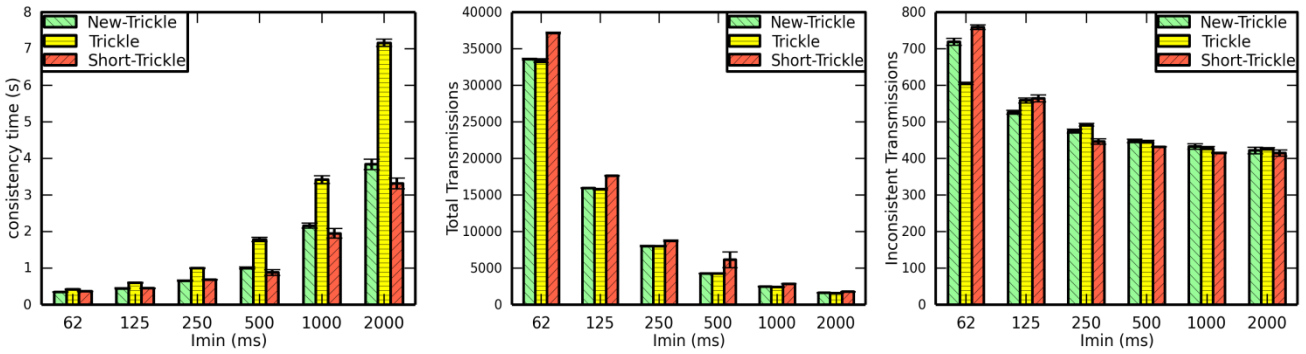
7.3.1. Main results

Figure 15 presents the consistency time, the total transmissions and the number of inconsistent packets registered by New-Trickle, Trickle and Short-Trickle when varying the minimum interval size, the redundancy constant and node transmission power.

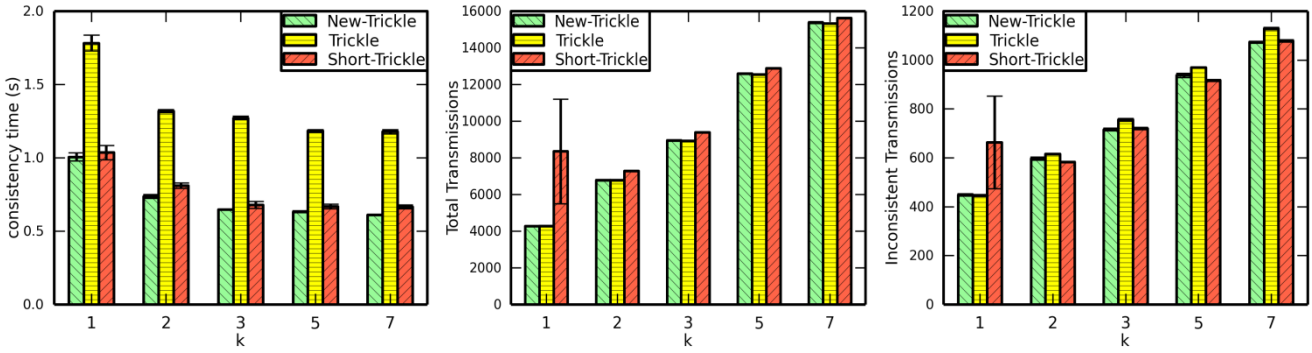
Figure 15 (a) depicts New-Trickle's performance when varying $Imin$. As can be seen from these graphs, New-Trickle achieved network consistency faster than Trickle, while approximately generating a similar number of transmissions. The small additional cost is due to the transmit-listen interplay discussed earlier. Note that even in this very irregular faulty network New-Trickle's propagation speed is less affected by the value of $Imin$. Concerning the number of generated inconsistent transmissions and similarly to previous results, New-Trickle generated fewer packets in the $Imin$ interval when the interval size was greater than $125\,ms$. In an $Imin = 62\,ms$, Trickle transmitted fewer packets, since it explicitly prevented some nodes form contending and hence minimised the chances for collisions and hidden terminals.

When varying k, as can be observed from Figure 15 (b), New-Trickle provided the best of both Trickle and Short-Trickle, even in this faulty irregular network experiencing heavy inconsistent traffic. Thus, it propagated new updates as quickly as Short-Trickle, which is about twice faster than Trickle, at a similar transmission cost. When considering the number of transmissions in the $Imin$ interval, New-Trickle generated less overhead than Trickle, as expected.

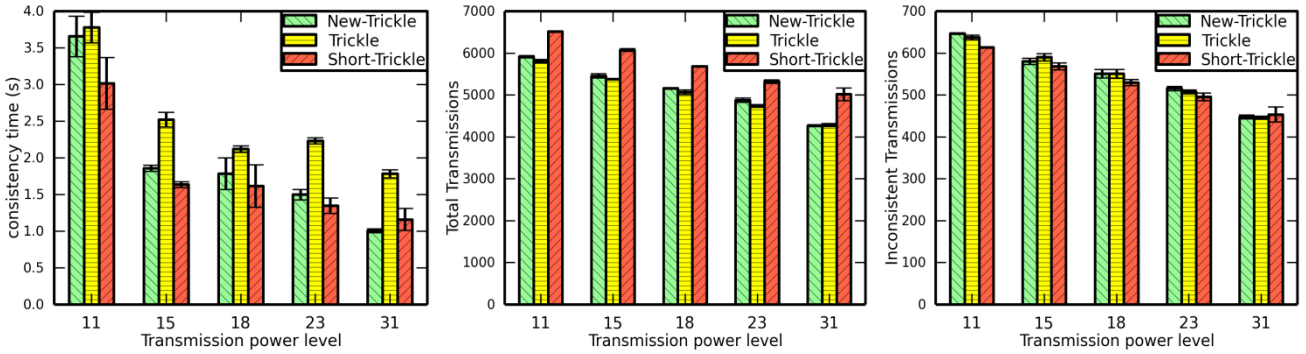
Finally, the third row of graphs depicted in Figure 15 (c) shows the performance of the evaluated protocols when varying transmission power. Varying transmission power plays a double role; it changes both the density and the success rate of the network. As can be seen from Figure 15 (c), New-Trickle propagated about twice faster than Trickle, even in a quite lossy, less connected network (power level 11 at -10 dBm). As expected and for the reasons disused earlier, New-Trickle's consistency time approached that of Short-Trickle, with a cost similar to that of Trickle.



(a) Varying *Imin*, transmission power = 0 dBm (level 31)



(b) Varying k, transmission power = 0 dBm (level 31)



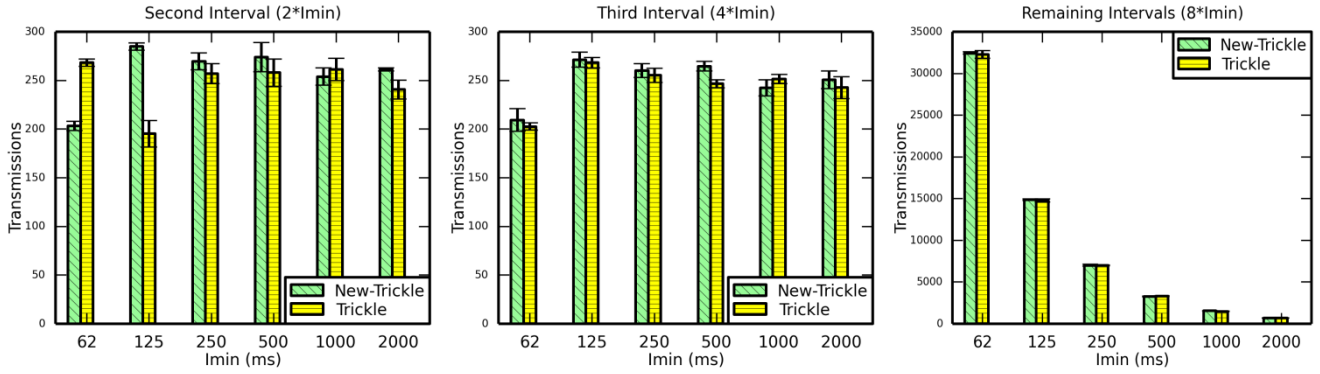
(c) Varying the transmission power

Figure 15 Performance evolution in the Indriya testbed

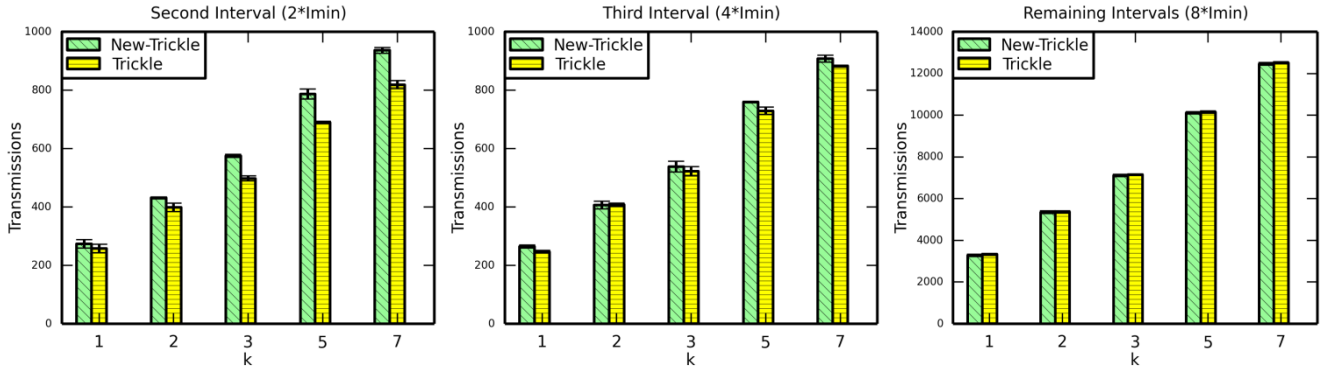
7.3.2. Quantifying the additional cost

In this subsection we quantify the additional cost observed in the above testbed results. Thus, we plot in Figure 16 the number of transmission generated by Trickle and New-Trickle in the second, third and remaining intervals.

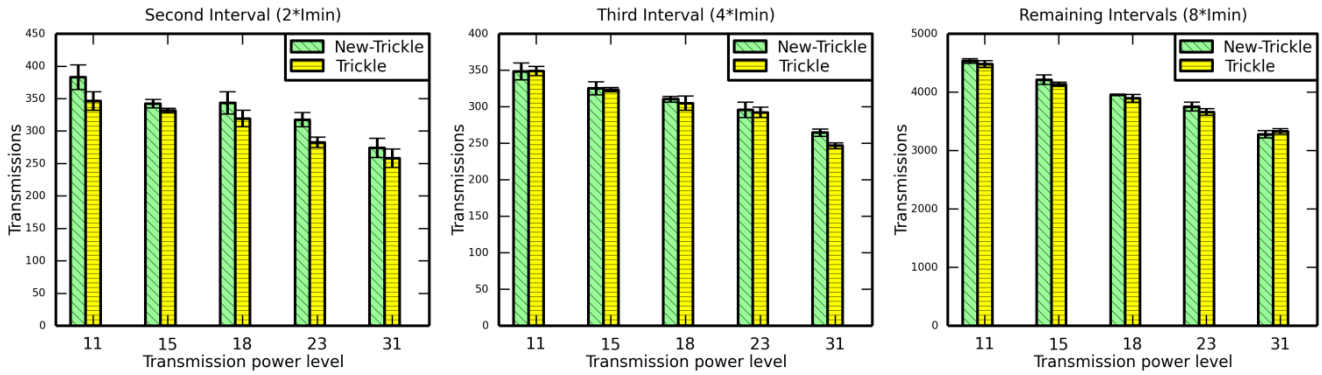
As can be seen for the graphs shown in Figure 16, the additional cost observed in New-Trickle disappears as early as the third interval under all the varied parameters, which confirms the simulation results. In the second and third intervals, and although Trickle sends fewer packets because of the transmit-listen interplay, the difference between the costs of the two protocols is very small. This is due mainly to the moderate density of the network. Finally, it should be noted that in an *Imin* of 62 ms, New-Trickle generated fewer packets in the second interval. This can be explained by the fact of Trickle generating fewer packets in the Imin interval (Figure 15 (c)), which reduced the benefits of the transmit-listen interplay, making Trickle transmit more in the second interval.



(a) Varying *Imin*, transmission power = 0 dBm (level 31)



(b) Varying k, transmission power = 0 dBm (level 31)



(c) Varying the transmission power

Figure 16 quantifying the additional cost in the Indriya testbed

7.4. Discussion summary

From the results discussed above, it can be concluded that New-Trickle can outperform Trickle in most of the key aspects. However, by trying to distribute the transmission load equitably, New-trickle can generate more transmissions in an *Imin* interval. Such is the case in dense networks adopting smaller values of *Imin*. This cost is mainly related to the efficiency of CSMA-based access strategies and timer-based suppression mechanisms. Note, however, that New-Trickle, however, exploits this cost to outperform Trickle in sparse and lossy networks as shown in Appendix A. To further illustrate this point, we carried out challenging simulations in a 4 neighbours/node deployment with every node undergoing a loss rate of 90 %. We repeated, 5 times, each experiment in which the seed node generated 10 inconsistent packets every 60 seconds. The results which are depicted in appendix A, show that New-Trickle can better cope with severe conditions. Finally, as we are particularly interested in the impact of bigger *Imin* values on Trickle and New-Trickle performance, we carried out other experiments, presented in appendix D, to show how the two algorithms behave when varying *Imin* between 4 and 20 seconds.

8. New-Trickle's Impact

Having demonstrated the performance of the New-Trickle algorithm, we briefly describe in this section its impact on the various Trickle-based use-cases and discuss new expected use-cases for real-time applications.

Overall, all Trickle-based applications can benefit from New-Trickle by modifying only a single line of code. Here we focus on the IETF standards RPL [5] and MPL [7].

RPL relies on Trickle for controlling the frequency of DIO (DODAG Information Object) messages, which constitute the basic building block of the DODAG (Destination Oriented Directed Acyclic Graph). Therefore, Trickle plays an important role in the convergence time and stability of RPL networks. In order to maintain a low consistency time, $Imin$ is recommended to equal 8 ms and k is conservatively proposed to equal 10. New-Trickle can offer faster consistency times even when using small Imin values, which helps to speed-up RPL's convergence. More importantly as New-Trickle propagation is loosely coupled with the value of Imin, choosing a big value of Imin can still allow a low RPL convergence time while minimizing the generated cost. In addition, as RPL uses big k values, the small additional cost observed in New-Trickle for equitably distributing the load can be compensated by the benefit of lower cost in the $Imin$ interval. Finally, New-Trickle may also prompt a new discussion of RPL's Trickle parameters and allow for a smaller value of k in order to minimize the cost of building and maintaining the DODAG.

On the other hand, MPL uses Trickle in both its proactive and reactive modes in order to achieve reliable multicast forwarding in LLNs. New-Trickle allows MPL to deliver multicast packets much faster. For instance, MPL's proactive mode recommends a default configuration of $k = 1$ and $Imax = Imin$, which invites most of the criticisms discussed in section 2.2, especially those concerning load distribution. Therefore, New-Trickle allows MPL's proactive mode to provide a balanced load distribution and, more importantly, avoids preventing some nodes from transmitting. It should be noted that MPL's proactive mode prevents this issue from becoming infinite by stopping a Trickle data timer after a specific number of expirations, recommended to equal 3. In addition, the optional proactive mode is backed by the reactive mode. Finally, New-Trickle can help MPL's reactive mode to achieve a better load distribution and a faster inconsistency resolution time.

Finally, by providing fast propagation times which are loosely-coupled to the value of $Imin$, New-Trickle can open doors for new real-time use cases of The Trickle algorithm.

9. Related Works

Trickle is becoming a basic network primitive in LLNs and is being deployed in many applications, ranging from routing control to data dissemination and distributed service/resource discovery. We discuss in this section related works trying to alter Trickle's behaviour. These works can be divided into two categories: those trying to tweak Trickle's behaviour in specific uses-cases, such as in RPL and MPL, and those studying Trickle's behaviour in the generic case, similar to our study.

As described earlier, RPL deploys Trickle to schedule DIO transmissions and, hence, dictates the consistency of the DODAG over time. This motivated researchers to study Trickle in order to predict the performance of RPL networks. For instance, [16] tries to make Trickle fair to all RPL nodes. Thus, it proposes to bias the uniform choice of transmission times by giving the nodes that sent fewer packets in the past more chances to transmit in the future. The authors of [17] observe the effect of non-synchronised Trickle intervals on RPL's generated control traffic. To tackle this effect, the authors proposed a readjustment of trickle intervals in order to gradually re-establish the steady state. The authors showed that the proposed scheme can fall back to Trickle in a worst case.

MPL describes a way of using Trickle to realise reliable multicast routing in LLNs. To this end, MPL introduces a fourth Trickle parameter, which allows stopping (destroying) a Trickle timer after a specific number of expirations. This allows efficient usage of the limited resources on constrained nodes since MPL's proactive mode requires for a node to create and manage a Trickle timer per packet. Note also that MPL uses Trickle to manage multiple data items in both proactive and reactive modes. Thus, it deploys, for the proactive mode, parallel Trickle approaches similar to the ones applied in [11][18]. For the reactive mode, MPL deploys serial Trickle approaches [19].

Other works focusing on analytically modelling Trickle's behaviour in order to predict its performance in generic cases are reported in [20][21][22]. These works try to provide mathematical tools that can analytically predict the message count

and the propagation of Trickle. For instance, a detailed analytical study of Trickle's behaviour is reported in [22], where the message count and propagation time of Trickle are analytically modelled as a function of a generalised listen-only period. However, while these models can help understand Trickle dynamics, they assume simplistic, lossless and regular network deployments. In addition, such models neither consider realistic radio propagation patterns, nor model contentions and collisions, which is an oversimplification of LLN dynamics.

10. Conclusion

In this report, we introduced, discussed and evaluated New-Trickle. Results showed important performance improvements, especially in the consistency time, compared to Trickle, while at the same time preserving Trickle's scalability. Nevertheless, many enhancements can be added to New-Trickle. For instance, New-Trickle works on the assumption that an inconsistent transmission is received simultaneously by the nodes that can hear it. While, this assumption seems reasonable in many cases, it can be challenged by some radio duty cycling protocols which change the semantics of broadcasts, such as ContikiMAC [23]. Thus, a broadcast transmission in ContikiMAC results in the sender repeating the same data over the whole Channel Check Interval (CCI). This is so in order to ensure that all receivers wake up and hence capture one of the repeatedly transmitted data. This means that nodes might not receive an inconsistent transmission simultaneously. As all the receivers will get the transmitted data during CCI, imposing a listen-only period of CCI length in an $Imin$-interval can solve this issue. However, this workaround might not be as efficient as required. In addition, transmission and reception energy profiles are totally different in duty-cycled networks, with a transmission costing over 10 times the cost of a reception [24]. This makes the communication load more costly for senders than for receivers. These issues are being currently investigated by the authors, and they are the first in the list of our works regarding Trickle. Finally, it should be noted that although we have provided a great deal of experimental evidence under Contiki OS, more evaluations and tests are required to fully understand the behaviour of New-Trickle and demonstrate that it does not violate Trickle assumptions in the most severe conditions.

Acknowledgments

The authors would like to acknowledge the insightful and fruitful discussions with Philip Levis which helped to shape this report.

References

[1] "10 Emerging Technologies That Will Change the World - MIT Technology Review." [Online]. Available: http://www2.technologyreview.com/featured-story/401775/10-emerging-technologies-that-will-change-the/3/. [Accessed: 20-Nov-2013].

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[3] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," in *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI*, 2004, pp. 15–28.

[4] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, "RFC 6206: The trickle algorithm," *IETF*, 2011.

[5] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. P. Vasseur, and R. Alexander, "RFC 6550: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," Mar. 2012.

[6] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, 2009, pp. 1–14.

[7] J. Hui and R. Kelsey, "Multicast Protocol for Low power and Lossy Networks (MPL)," 2014.

[8] B. Djamaa, M. Richardson, N. Aouf, and B. Walters, "Towards efficient distributed service discovery in low-power and lossy networks," *Wirel. Netw.*, vol. 20, no. 8, pp. 2437–2453, Nov. 2014.

[9] A. Chlipala, J. Hui, and G. Tolle, "Deluge: data dissemination for network reprogramming at scale," *Univ. Calif. Berkeley Tech Rep*, 2004.

[10] K. Lin and P. Levis, "Data discovery and dissemination with dip," in *Proceedings of the 7th international conference on Information processing in sensor networks*, 2008, pp. 433–444.

[11] G. Tolle and D. Culler, "Design of an application-cooperative management system for wireless sensor networks," in *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, 2005, pp. 121–132.

[12] T. Dang, N. Bulusu, W.-C. Feng, and S. Park, "Dhv: A code consistency maintenance protocol for multi-hop wireless sensor networks," in *Wireless Sensor Networks*, Springer, 2009, pp. 327–342.

[13] P. Levis, E. Brewer, D. Culler, D. Gay, S. Madden, N. Patel, J. Polastre, S. Shenker, R. Szewczyk, and A. Woo, "The emergence of a networking primitive in wireless sensor networks," *Commun. ACM*, vol. 51, no. 7, pp. 99–106, 2008.

[14] IEEE Computer Society, LAN/MAN Standards Committee, Institute of Electrical and Electronics Engineers, and IEEE-SA Standards Board, *IEEE standard for information technology telecommunications and information exchange between systems--local and metropolitan area networks--specific requirements. Part 15.4, Part 15.4,*. New York, NY: Institute of Electrical and Electronics Engineers, 2006.

[15] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda, "Indriya: A low-cost, 3D wireless sensor network testbed," in *Testbeds and Research Infrastructure. Development of Networks and Communities*, Springer, 2012, pp. 302–316.

[16] C. Vallati and E. Mingozzi, "Trickle-F: Fair broadcast suppression to improve energy-efficient route formation with the RPL routing protocol," in *Sustainable Internet and ICT for Sustainability (SustainIT), 2013*, 2013, pp. 1–9.

[17] J. Eriksson and O. Gnawali, "Synchronizing Trickle Intervals."

[18] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 81–94.

[19] O. Gnawali, K.-Y. Jang, J. Paek, M. Vieira, R. Govindan, B. Greenstein, A. Joki, D. Estrin, and E. Kohler, "The tenet architecture for tiered sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, 2006, pp. 153–166.

[20] H. Kermajani, C. Gomez, and M. H. Arshad, "Modeling the Message Count of the Trickle Algorithm in a Steady-State, Static Wireless Sensor Network," *IEEE Commun. Lett.*, vol. 16, no. 12, pp. 1960–1963, Dec. 2012.

[21] M. Becker, K. Kuladinithi, and C. Görg, "Modelling and Simulating the Trickle Algorithm," in *Mobile Networks and Management*, Springer, 2012, pp. 135–144.

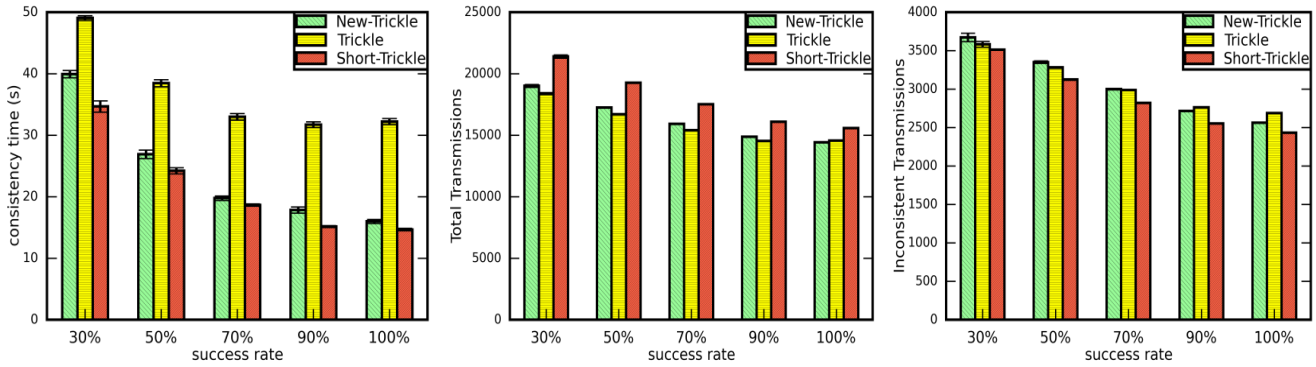
[22] T. M. M. Meyfroyt, "Modeling and analyzing the Trickle algorithm," MsC, 2013.

[23] A. Dunkels, "The contikimac radio duty cycling protocol," 2011.

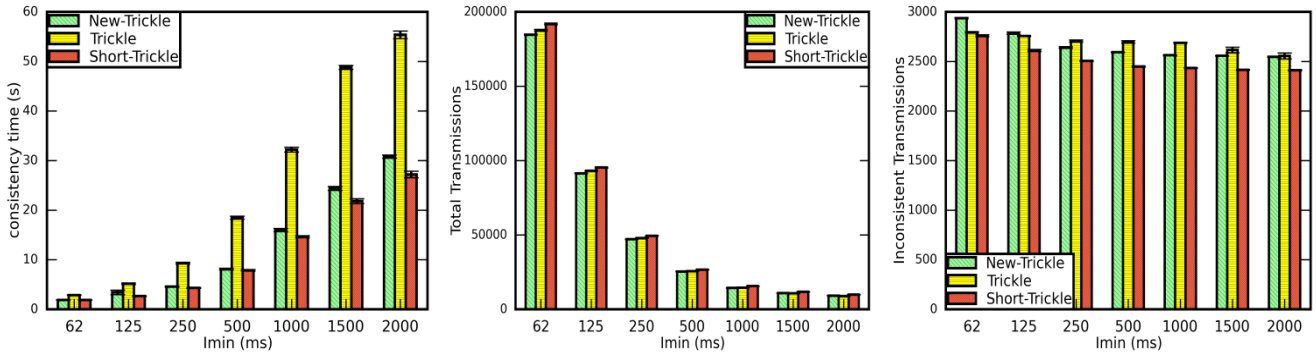
[24] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, "Powertrace: Network-level power profiling for low-power wireless networks," 2011.

Appendix A: Results from a 400-node sparse network deployed in a grid (Setup 2, 4 neighbour/node density)

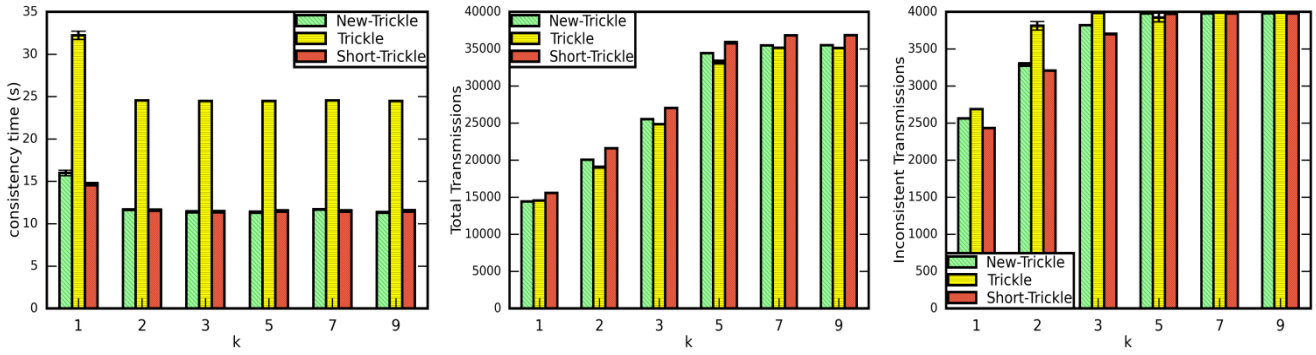
I. Main results



(a) Varying the physical success rate

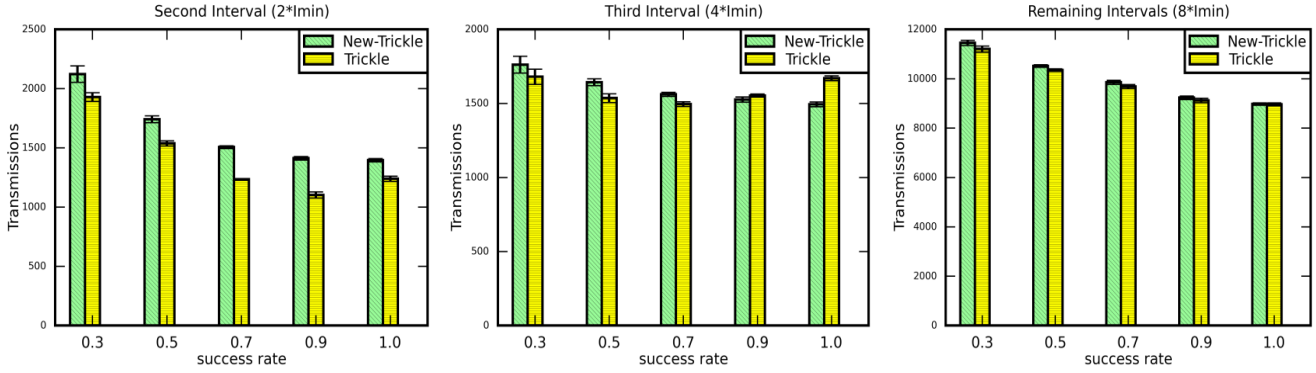


(b) Varying *Imin*, success rate = 100 %

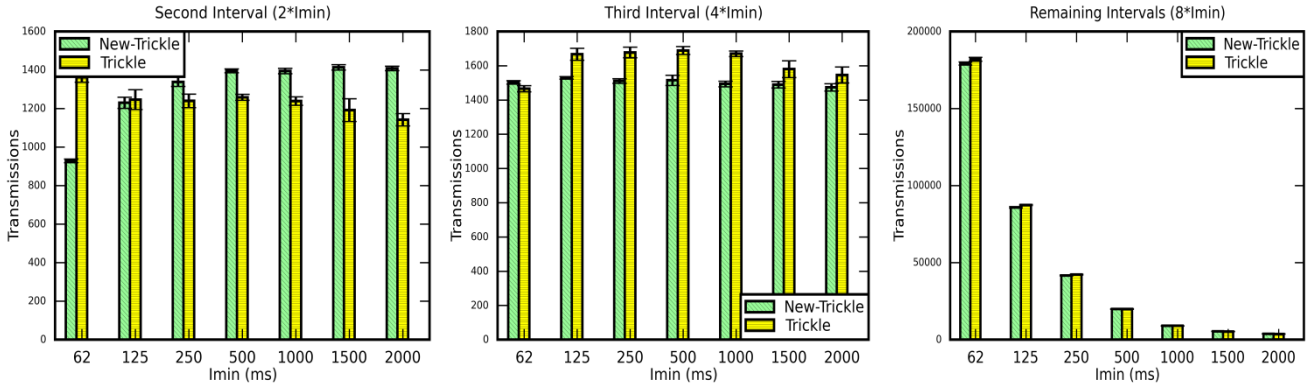


(c) Varying k, success rate = 100 %

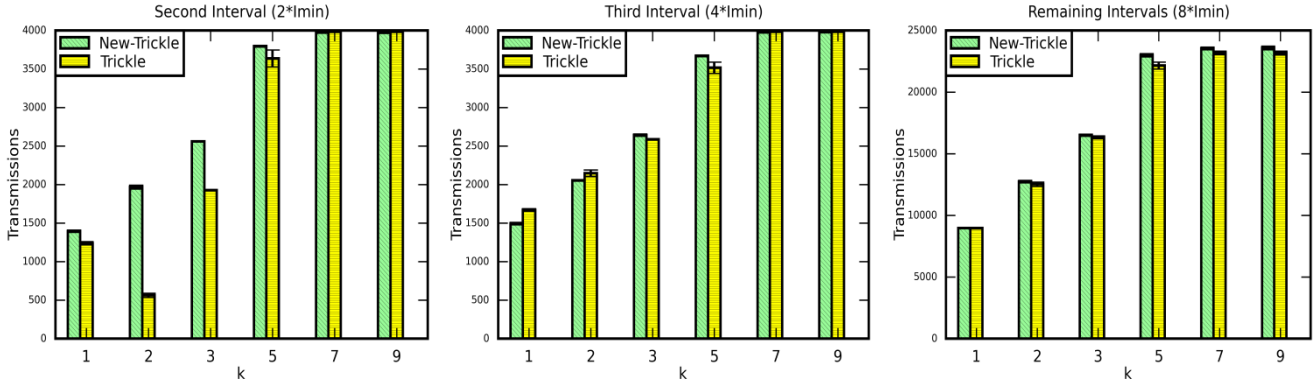
II. Quantifying the additional cost



(a) Varying physical success rate

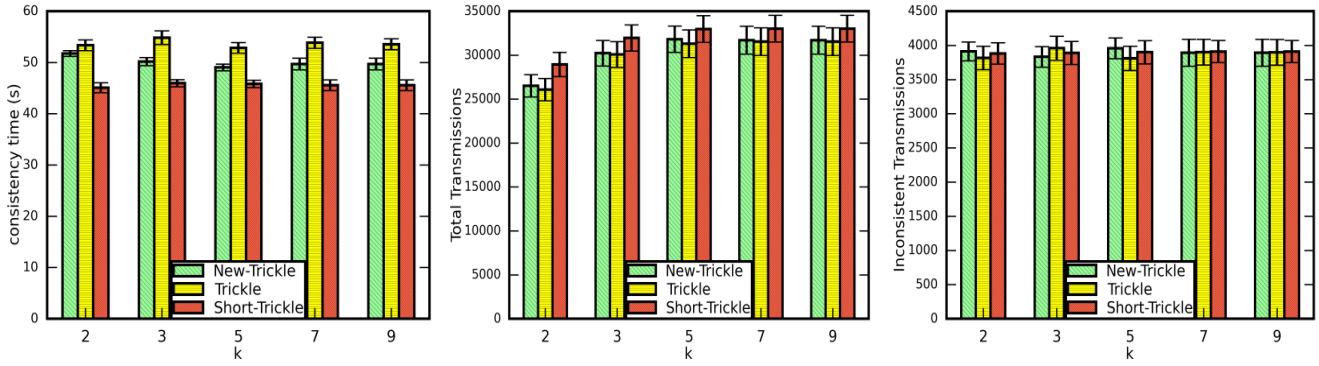


(b) Varying *Imin*, success rate = 100 %

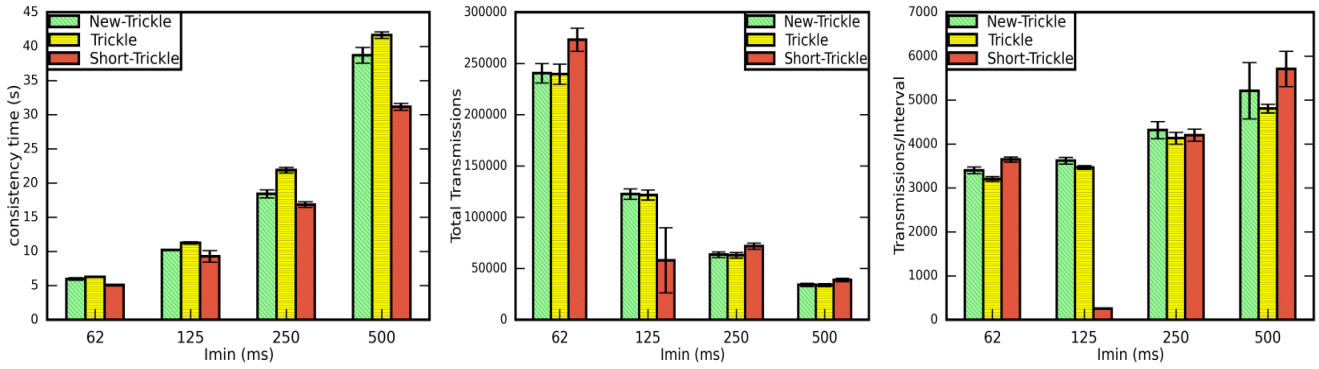


(a) Varying *k*. success rate = 100 %

III. Sever conditions



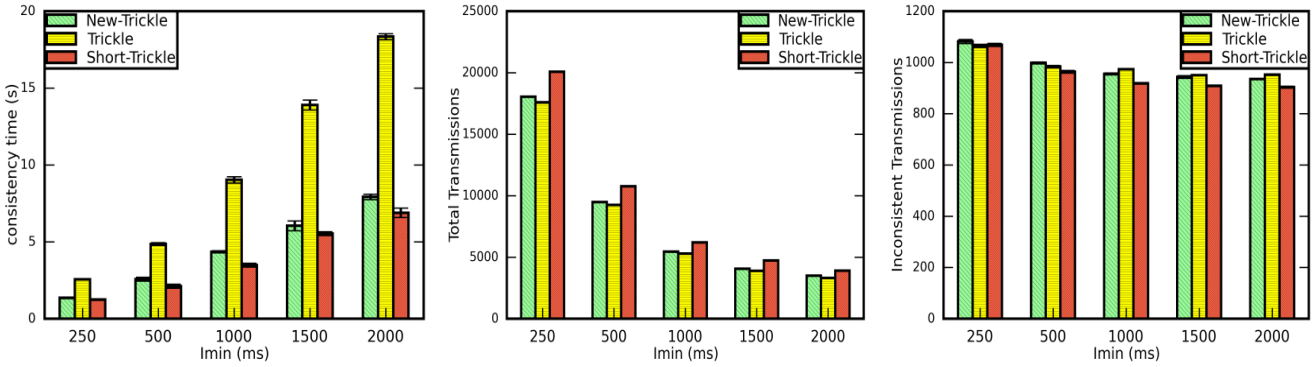
(a) Varying *k*, success rate = 10 %



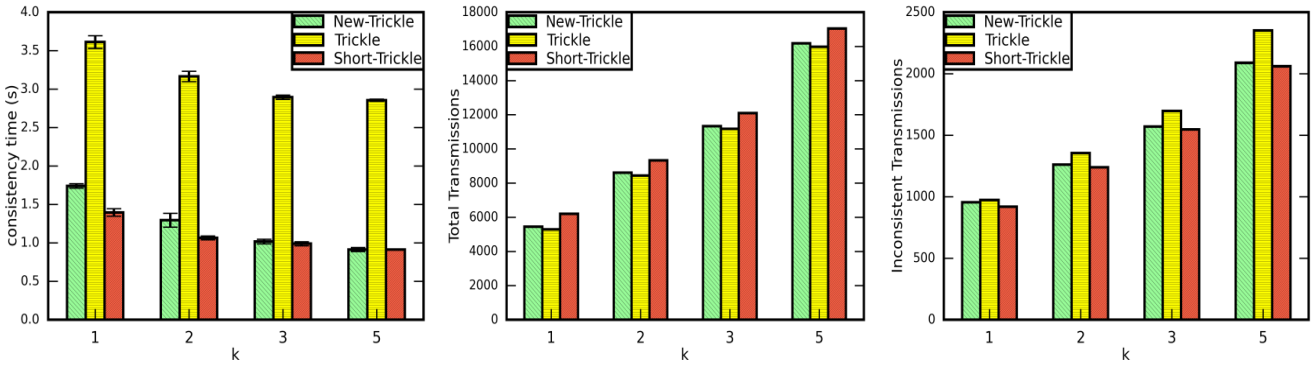
(b) Varying *Imin*, success rate = 10 %

Appendix B: 400-node uniformly distributed dense network (Setup 2, 35 neighbour/node average density)

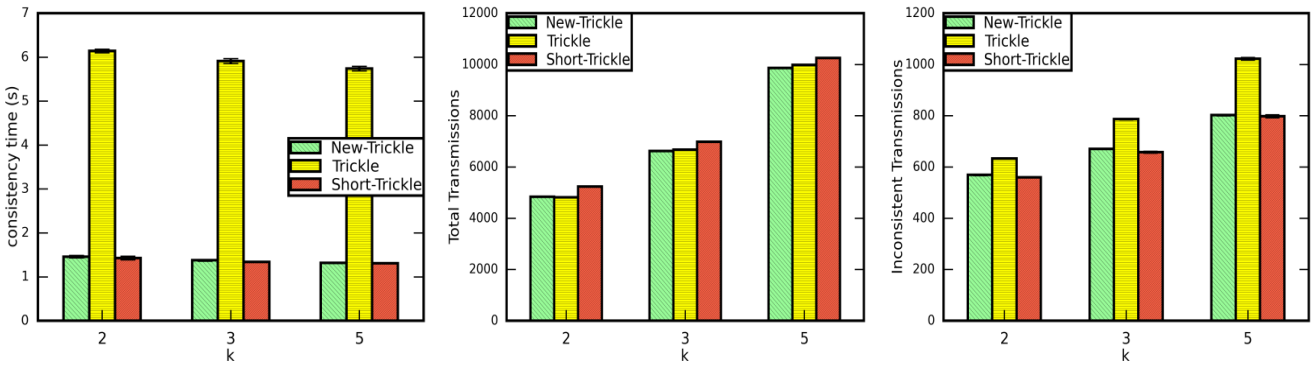
I. Main results



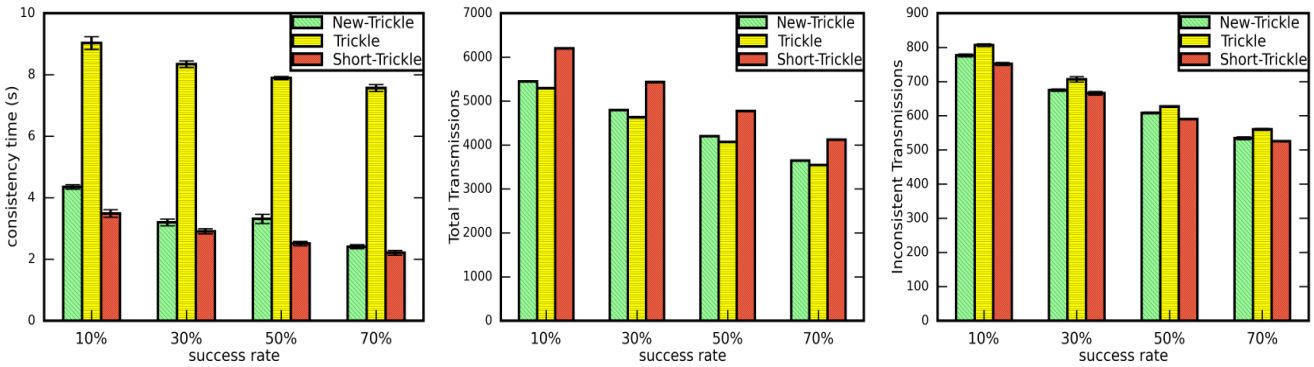
(a) Varying *Imin*, success rate = 10 %



(b) Varying k, success rate = 10 %



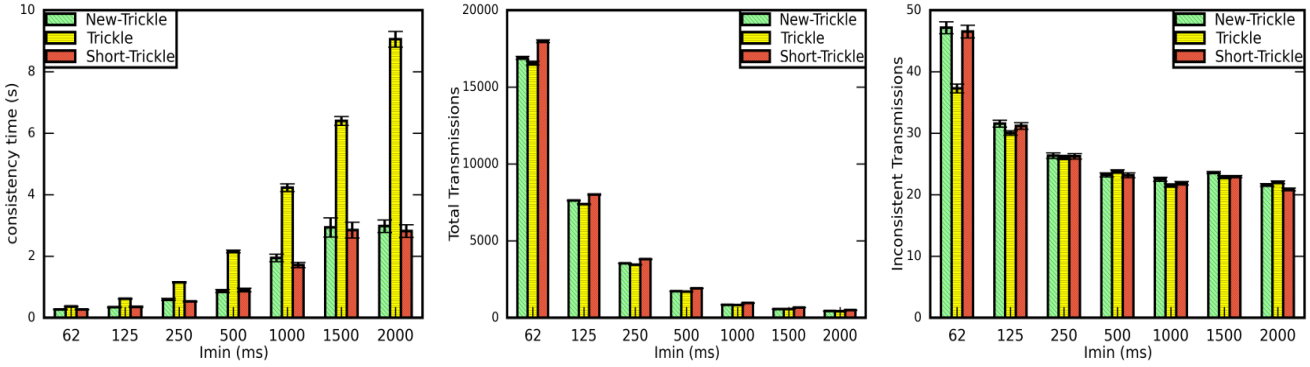
(c) Varying k, success rate = 100 %



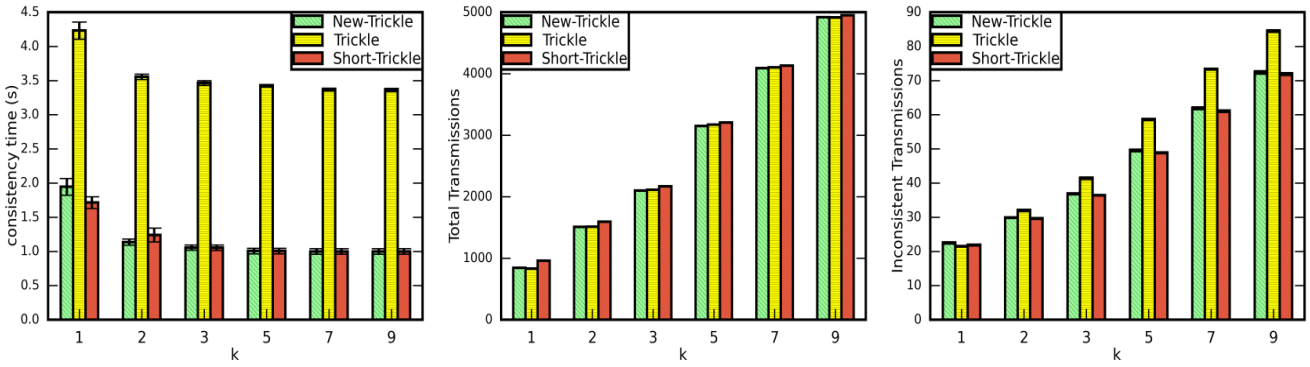
(d) Varying the physical success rate = 100 %

Appendix C: Results from a 100-node network deployed in a grid (Setup 1, average of 20 neighbour/node density)

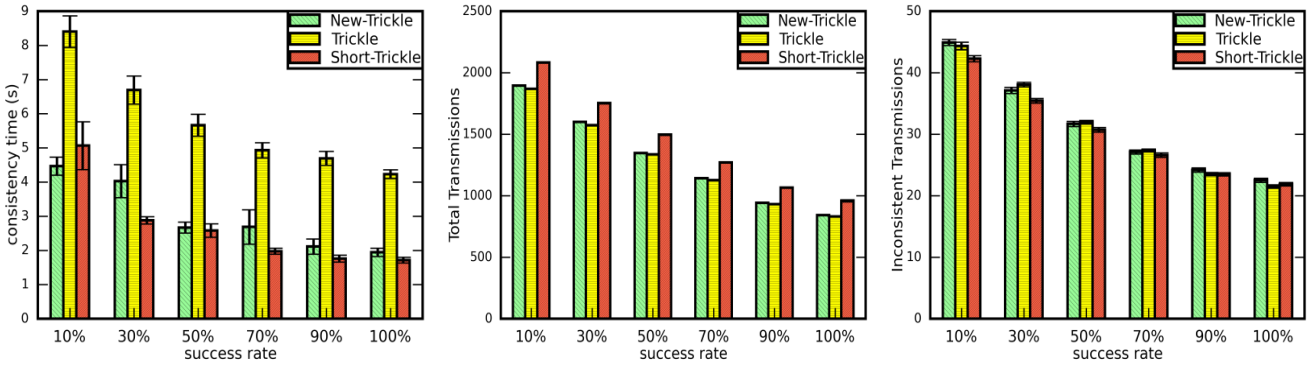
I. Main Results



(a) Varying *Imin*, success rate = 100 %



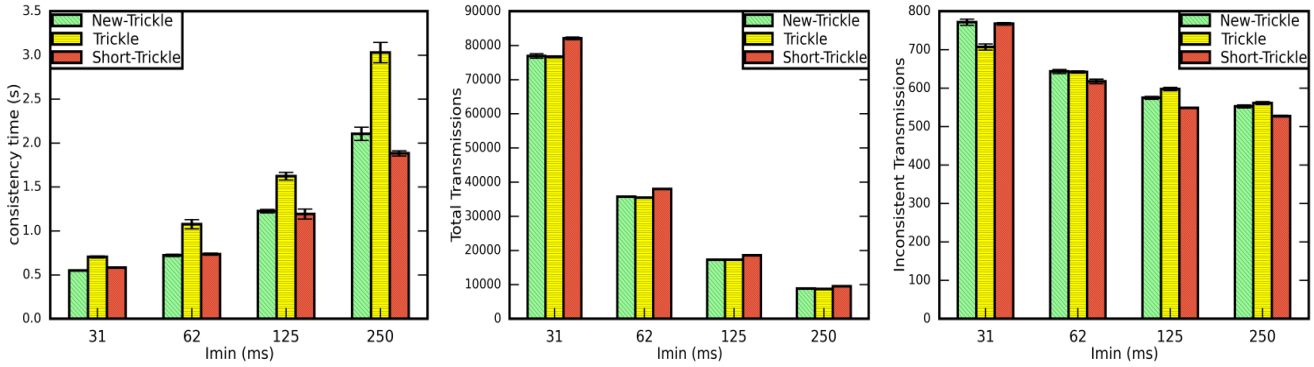
(b) Varying *k*, success rate = 100 %



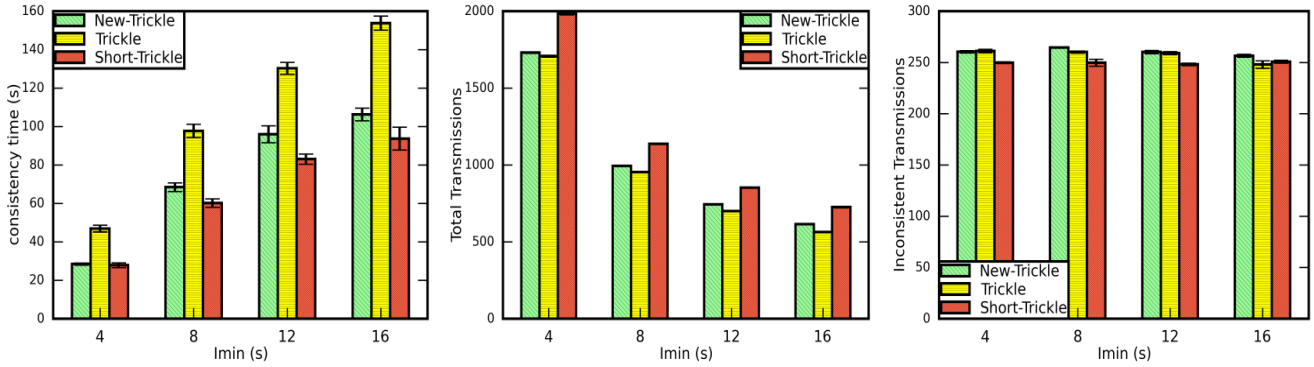
(c) Varying the physical success rate

Appendix D: Big and small Imin values, Setup 2

II. 100-node uniformly distributed sparse network (average density of 6 neighbour/node)

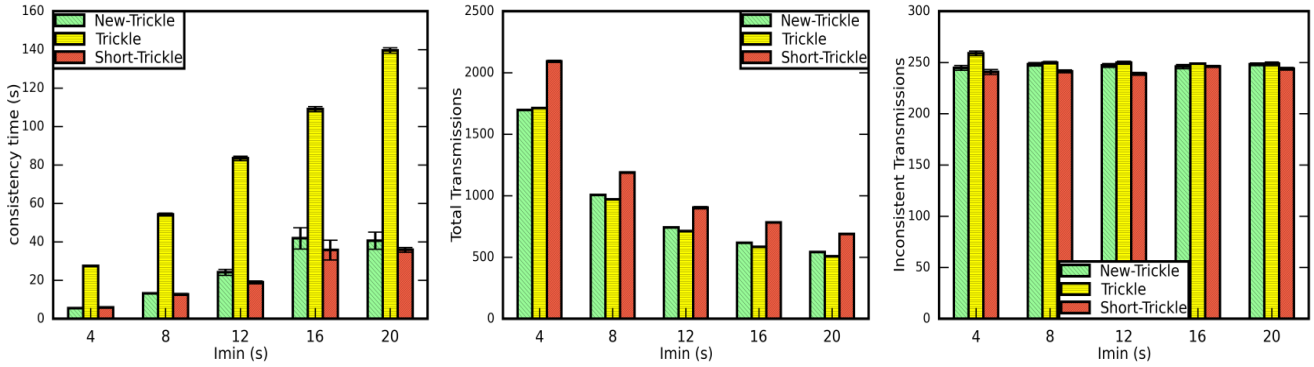


(a) Small *Imin*, success rate = 100 %



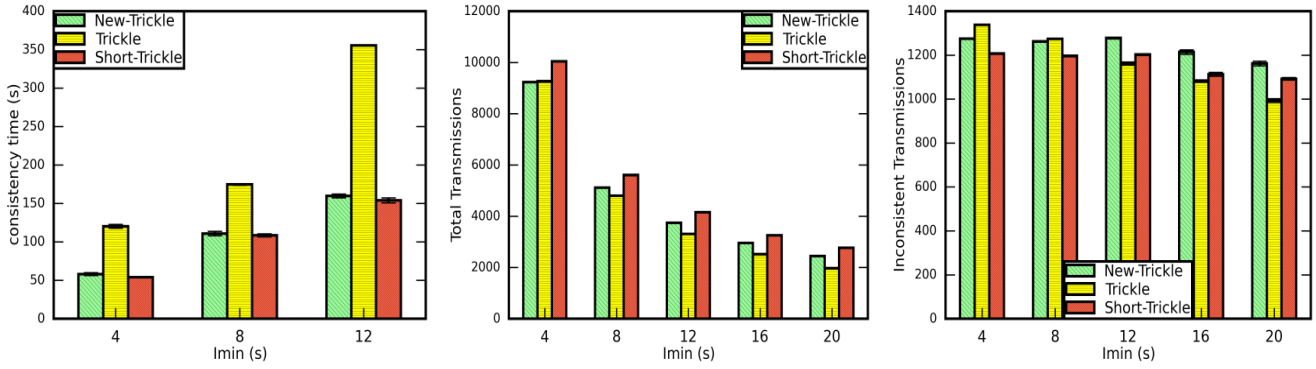
(b) Bigger *Imin* values, success rate = 100 %

III. 400-node uniformly distributed dense network (average density of 35 neighbour/node)



(a) Bigger *Imin* values, success rate = 100 %

IV. 400-node grid topology sparse network (average density of about 4 neighbour/node)



(a) Bigger *Imin* values, success rate = 100 %. (50% network consistency time)

Appendix E: Indriya testbed layout